VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

VYSOKORYCHLOSTNÍ FILTRACE SÍŤOVÉHO PROVOZU
HIGH-SPEED FILTRATION OF NETWORK TRAFFIC

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan Churý**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jan Hajný, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Jan Churý ***ID:*** 187010
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Vysokorychlostní filtrace síťového provozu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je nalézt a srovnat knihovny a nástroje pro vysokorychlostní filtraci síťového provozu. Výstupem práce bude softwarová implementace schopná filtrovat a distribuovat/agregovat síťový provoz na rychlosti 10 Gb/s. Implementace bude otestována pro různé typy provozu (velikosti paketů, množství relací, přenosové rychlosti) a různé pravidla filtrace (dle obsahu IP hlaviček, spojení). Výsledná implementace bude optimalizována pro min. 10 Gb/s linku.

DOPORUČENÁ LITERATURA:

[1] Debian - Documentation. Debian - Documentation [online]. 2016 [cit. 2016-09-12]. Dostupné z: https://www.debian.org/doc/index.en.html

[2] High-Speed Traffic Capture and Analysis Using Open-Source Software and Commodity Hardware [online]. [cit. 2016-09-12]. Dostupné z: http://luca.ntop.org/TMA_PhD_School_2011.pdf

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 24.5.2017

Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Anotace
Pro filtrování síťového provozu při vyšších přenosových rychlostech (například více než 1 Gbit/s), je v dnešní době k dispozici mnoho proprietárních hardwarových řešení. Ovšem existuje i několik projektů se svobodnou licencí, které se zaměřují na zpracování vysokorychlostních síťových dat na běžném hardwaru. Cílem práce je nalézt takovéto projekty, ověřit, zda existují filtrovací nástroje na nich založené, vyzkoušet filtraci 10Gbit/s síťového provozu a otestovat je pro různá nastavení filtrací. Výstupem práce by měla být implementace síťového filtru, který dokáže filtrovat provoz až do rychlosti 10 Gbit/s.
Klíčová slova: vysokorychlostní filtrace packetů, 10 Gbit/s, firewall, netmap, nftables, PF_RING, Linux

Abstract
For high-speed (e.g. more than 1 Gbit/s) filtration of network traffic there are available many of proprietary hardware solutions nowadays. But there are also a couple of free licensed projects that are specialized in high-speed packet processing on common hardware. The goal of thesis is to find such projects, verify that there are filtering tools based on these projects, try to filter 10Gbit/s network traffic by these tools and test them against various filtration settings. Implementation of packet filter that could be used for filtration of network traffic up to 10Gbit/s speed should be the output of this thesis.
Keywords: high-speed filtration of network traffic, 10 Gbit/s, firewall, netmap, nftables, PF_RING, Linux

CHURÝ, J. *Vysokorychlostní filtrace síťového provozu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017. 119 s. Vedoucí diplomové práce doc. Ing. Jan Hajný, Ph.D.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma „Vysokorychlostní filtrace síťového provozu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne 22. 5. 2017. .....

Bc. Jan Churý

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

Poděkování

Děkuji vedoucímu práce doc. Ing. Janu Hajnému, Ph.D. za veškerou pomoc, cenné rady, metodické vedení a čas, který mi při zpracovávání práce věnoval. Také bych chtěl poděkovat Ing. Zdeňku Martináskovi, Ph.D. a panu Janu Sedlákovi za konzultace a pomoc při přepojování experimentální sítě.

V Brně dne 22. 5. 2017. .....

Bc. Jan Churý

Obsah

1. Úvod

Filtrování síťového provozu je jednou ze základních bezpečnostních technik v počítačových sítích. Zatímco v dřívějších dobách, kdy rychlosti připojení dosahovaly několika stovek kilobitů až jednotek megabitů za sekundu, nebyl problém s filtrováním na běžně dostupném hardwaru a softwaru, v dnešní době, kdy připojení nezřídka dosahují rychlosti několika desítek až stovek megabitů za sekundu (ale samozřejmě i více), začíná být stále palčivější otázkou, jaký hardware (a případně software) zvolit tak, aby se samotný firewall nestal úzkým hrdlem sítě. Většinou se to řeší použitím nějakého proprietárního hardwarového zařízení, které obsahuje různé speciální čipy (např. ASIC, FPGA atd.) a pro které výrobce garantuje určitou datovou propustnost. Bohužel je takovéto řešení většinou drahé a nemusí poskytovat dostatečnou flexibilitu, kterou zákazník žádá. Problém s běžně dostupným hardwarem a softwarem je ten, že nejsou pro tento typ úlohy optimalizovány. Pro příklad je možné uvést jádro Linux, jímž se tato práce zabývá. Linux je univerzální jádro operačního systému, jenž může běžet na hardwaru velikosti USB flash disku nebo také sálového superpočítače. Může sloužit od jednoúčelového sběru dat ze senzoru přes ovládání mobilního telefonu či osobního počítače až k provozu serverových aplikací nebo ke zpracování náročných výpočtů. A samozřejmě k filtrování síťového provozu. Ovšem na žádnou tuto činnost není úzce profilován, a proto ji dokáže provádět s určitými limity. Díky otevřenosti zdrojových kódů a dobré dokumentaci však lze linuxové jádro upravit tak, aby určitý typ úlohy zpracovávalo lépe než jádro neupravené. To je i hlavním úkolem této práce: nalézt, vyzkoušet a porovnat nástroje a knihovny, které za pomoci Linuxu dokáží filtrovat procházející síťový provoz, a to nejlépe až do datové propustnosti 10 Gbit/s. Tato činnost by měla být ideálně zcela transparentní, tedy pracovat na linkové vrstvě (L2) referenčního modelu ISO OSI.

První část práce se věnuje obecně firewallům. Vysvětluje, co to je firewall, do jakých skupin a podle jakých kritérií lze firewally dělit, i to, do kterých částí sítě se firewall obvykle nasazuje.

Další část pojednává o možnostech filtrace síťového provozu v Linuxu. Zaměřuje se hlavně na nástroje a knihovny, které slouží k vysokorychlostnímu zpracování síťových dat.

Poslední částí práce je samotné praktické zpracování tématu, tedy představení testovacího prostředí a testovacích metod, odůvodnění výběru nástrojů či knihoven, které budou v testovacím prostředí odzkoušeny, a samozřejmě uvedení naměřených výsledků a jejich vyhodnocení.

Ačkoliv je v zadání napsáno, že by výsledná implementace měla být optimalizována minimálně pro 10Gbit/s linku, v průběhu zpracovávání práce byla po určitý čas dostupná infrastruktura „jen“ pro rychlost 10 Gbit/s, a tudíž mohla být vyzkoušena filtrace maximálně na této rychlosti. Pro vyšší přenosové rychlosti nebylo k dispozici potřebné vybavení. Obdobně to platí pro agregaci síťových rozhraní, kdy bylo možné vyzkoušet agregaci jen pro dvě 1Gbit/s rozhraní (tedy do propustnosti 2 Gbit/s).

2. Firewall

Jako firewall označujeme hardware nebo software, který monitoruje vstupní i výstupní provoz ze síťových rozhraní, a na základě množiny bezpečnostních pravidel rozhoduje, zda daný provoz má být propuštěn (např. ke zpracování v aplikační vrstvě či odeslán ze síťového rozhraní) nebo zablokován. O tom, zda má být každý jednotlivý packet povolen či zahozen, se zpravidla rozhoduje na základě obsahu hlaviček linkové, síťové či transportní vrstvy ISO OSI modelu, ale používají se i firewally, které dokáží kontrolovat síťový provoz na základě obsahu dat aplikační vrstvy. Kromě samotného filtrování nabízí firewally i jiné funkce – např. překlad síťových adres (NAT), zakončení tunelů virtuálních privátních sítí (VPN), směrování atd.

Existují různé druhy firewallů, které můžeme rozdělit do mnoha skupin. Většinou však nelze říci, že by nějaký konkrétní firewall patřil pouze do jedné skupiny. Častější jsou firewally, které dokáží fungovat ve více režimech a můžeme je tedy zařadit do více skupin.

2.1. Rozdělení firewallů

2.1.1. Podle umístění v síti

Síťový (network-based) firewall

Jedná se o síťové zařízení, které se připojuje na rozhraní dvou a více sítí. Tyto sítě, v mnoha případech, spravují různé osoby, proto je nutné nedůvěřovat veškerému provozu, který prochází mezi sítěmi, a raději jej filtrovat podle bezpečnostních politik chráněné sítě. Jak bylo naznačeno, minimálně jedna síť je důvěryhodná, a tedy firewallem chráněná, zbylá síť/sítě jsou nedůvěryhodné (např. Internet). Z textu vyplývá, že síťový firewall se používá k ochraně celé sítě, tedy více koncových stanic (klientských počítačů či serverů) a mezilehlých síťových prvků (směrovačů, přepínačů). Síťový firewall obvykle funguje i jako hraniční směrovač důvěryhodné sítě.

Osobní (host-based) firewall

Jak již název napovídá, jedná se o firewall, který slouží k ochraně jedné konkrétní koncové stanice (klientského počítače či serveru). Standardně bývá tento typ firewallu součástí operačního systému. Osobní firewall slouží k ochraně operačního systému a aplikací na něm běžících. Díky možnosti filtrování i odchozího provozu ze stanice lze ochránit i další zařízení v síti v případě, že by došlo ke zkompromitování uživatelské aplikace.

2.1.2. Podle principu fungování

Packetový filtr

Jedná se o nejjednodušší typ firewallu. Jeho činnost je založena na porovnávání každého jednotlivého packetu vůči bezpečnostní politice, kterou zde představuje filtrovací tabulka. Ta je tvořena sloupci pro zdrojovou a cílovou síťovou adresu daného packetu, zdrojový a cílový port aplikace a většinou také obsahuje sloupec pro typ protokolu, který je přenášen v datové části internetového protokolu. Nezbytnou součástí tabulky je i sloupec

pro zapsání akce, která má být s vyhovujícím packetem provedena – povolení či zahození. Algoritmus samotného firewallu se tedy stará o zjištění údajů o adresách, portech a protokolu z každého příchozího (či odchozího) packetu a porovnává tyto údaje vůči pravidlům ve filtrovací tabulce. Ta je procházena odshora dolů a podle prvního pravidla, které zpracovávanému packetu vyhoví, je s packetem naloženo (podle informace ze sloupce akce). Proto je potřeba do tabulky vkládat pravidla od specifických k obecnějším. V případě, že packet nevyhoví ani jednomu pravidlu, bývá na konci tabulky uvedeno obecné pravidlo, které každý packet povolí nebo zahodí.

Důležitou vlastností tohoto firewallu je to, že každý packet je vyhodnocován zvlášť bez jakýchkoliv vazeb na dřívější komunikaci. To s sebou přináší výhodu celkem rychlého zpracování, ovšem jedná se o velice jednoduché filtrování, kdy firewall nemá možnost kontrolovat, zda komunikace probíhá ve správném pořadí a v souladu s pravidly, které jsou definovány ve standardech jednotlivých protokolů (např. správné sestavení TCP komunikace na základě třícestného vyjednávání).

Packetový filtr se stavovou inspekcí

Packetový filtr se stavovou inspekcí je rozšířenou verzí obyčejného packetového filtru. Kromě filtrovací tabulky si vede i tabulku povolených spojení. V té si zaznamenává stav spojení, která přes firewall prochází. Přijde-li na firewall packet, je obvykle nejdříve porovnáván vůči tabulce spojení. Pokud je spojení, ke kterému packet patří, v tabulce nalezeno, může být provedena další protokolová kontrola (např. správná sekvenční a potvrzovací čísla v hlavičce TCP) a aktualizace stavu ve stavové tabulce. V případě, že i těmto následným kontrolám packet vyhoví, je firewallem akceptován. Pokud záznam v tabulce spojení není, musí být packet porovnán vůči filtrovací tabulce. Jestliže packet vyhoví pravidlům z filtrovací tabulky, je vytvořen nový záznam v tabulce spojení, na základě kterého bude kontrolována a propouštěna následná komunikace. Ačkoliv by se mohlo zdát, že stavová inspekce má význam pouze u stavových protokolů (nejznámější zástupce je zřejmě TCP), běžně se využívá i u protokolů bezstavových (např. UDP, ICMP). V tom případě se ke každému takovému spojení přiřadí časovač, po jehož vypršení se spojení z tabulky smaže. Pokud by následovala další komunikace daného spojení, musela by se opět ověřovat vůči filtrovací tabulce.

Výhodou tohoto typu firewallu je schopnost rozpoznávat a rozumět určitým protokolům a také vyšší rychlost zpracování síťového provozu díky tomu, že známá a již povolená spojení se vyhledávají rychleji v tabulce spojení, než kdyby se měla kontrolovat vůči filtrovací tabulce [1].

Aplikační brána (příp. proxy firewall)

Aplikační brána filtruje procházející provoz až v aplikační vrstvě modelu ISO OSI. Algoritmus firewallu postupně načítá procházející packety tak, aby tvořily kompletní zprávu aplikace (v případě, že je zpráva rozdělená do více packetů) a nad ní pak provádí kontrolu. Přitom nekontroluje pouze hlavičky aplikačních protokolů, ale často analyzuje i samotná aplikační data. Tím lze objevit škodlivý kód, který je možné zahodit dříve, než dorazí na cílovou stanici. Samotná komunikace neprobíhá přímo mezi klientem v chráněné síti a serverem v nedůvěryhodné síti, jako tomu bylo u packetových filtrů, ale zprostředkovává ji právě aplikační brána. Klient tedy otevře nové spojení na aplikační bránu a předá jí svůj

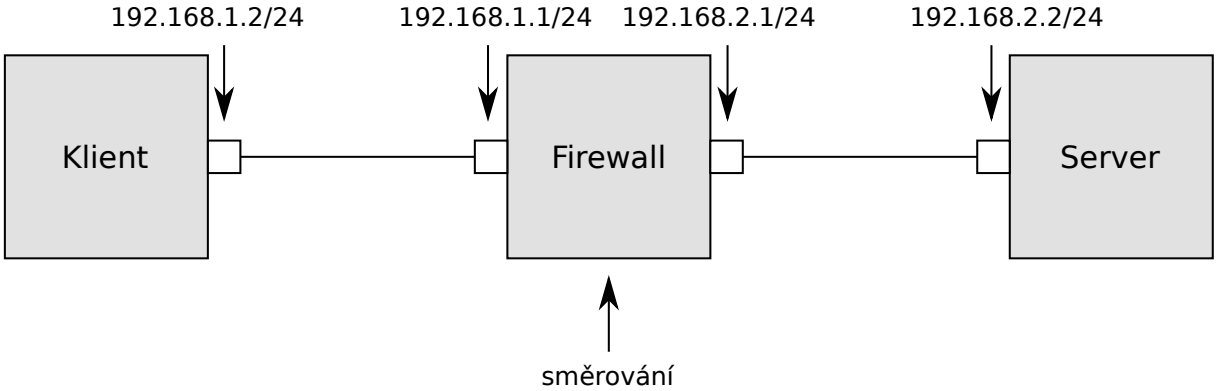
požadavek na komunikaci. Brána otevře spojení s cílovou stanicí z přijatého požadavku a přepošle jí požadavek od klienta. Odpověď serveru brána přijme a předá ji filtrovací aplikaci, která nad ní provede kontrolu. Pokud je vše v pořádku, je odpověď serveru přeposlána již otevřeným spojením na klienta.

Z popisu principu fungování aplikační brány je jasné, že výhodou tohoto typu firewallu bude vysoká ochrana chráněné sítě. Ovšem na druhou stranu, kvůli filtraci až na sedmé vrstvě ISO OSI modelu, je datová propustnost menší než u packetových filtrů, a navíc musí být klientská aplikace přizpůsobená ke komunikaci přes aplikační bránu.

2.1.3. Podle transparentnosti

Netransparentní firewall

V tomto režimu funguje firewall jako klasický směrovač. Kromě toho však směrovaný provoz i filtruje. Jako každý jiný směrovač tedy musí mít minimálně dvě fyzická či logická síťová rozhraní, přičemž každé z nich má přidělenou adresu z různých (pod)sítí (viz obrázek 2.1). Po přijetí packetu je tedy mimo samotné filtrace provedeno i směrovací rozhodnutí – tedy na základě směrovací tabulky je rozhodnuto, na které odchozí rozhraní bude packet předán. Díky tomu může být firewall použit pro zakončení VPN tunelů, být součástí domény směrovačů pro dynamické směrování nebo třeba dělat překlad síťových adres. Nevýhodou může být určitá možnost detekce firewallu v síti či možné problémy s adresováním při nasazení. Mluvíme-li obecně o firewallech, máme ve většině případů na mysli tento typ.

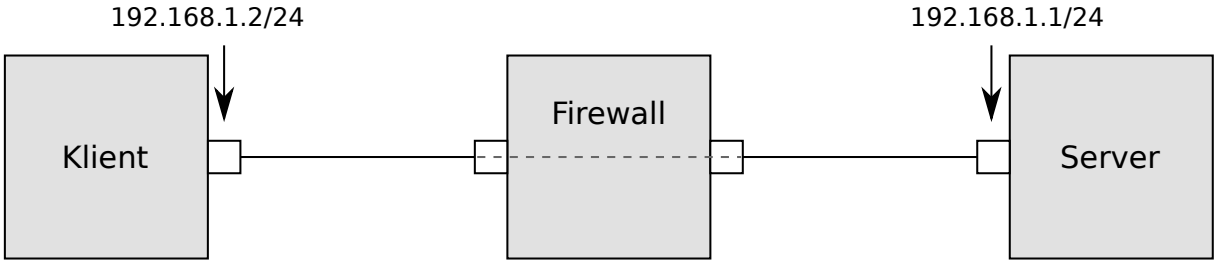


Obrázek 2.1: Schéma funkčnosti netransparentního firewallu.

Transparentní firewall

Firewall v tomto režimu se chová zcela transparentně pro veškerý IP provoz. Jeho činnost spočívá v příjmu rámce, jeho zkontrolování a v případě, že vyhoví nastaveným pravidlům, je předán na výstupní rozhraní (viz obrázek 2.2). Aby tento způsob předávání rámců fungoval, musí síťová rozhraní podporovat a být přepnuta do tzv. „promiskuitního módu“. Při příjmu rámce, který obsahuje jinou cílovou linkovou adresu než je linková adresa rozhraní, není v tomto módu rámec zahozen, ale je předán dále ke zpracování. Díky tomu, že se firewall nepodílí na směrování provozu, je v síti téměř nezjistitelný (někdy je označovaný jako „stealth firewall“), nerozděluje adresní prostor tam, kde je nasazen, a je možné ho použít k ochraně zařízení, které neobsahuje vlastní firewall, bez nutnosti

předadresace. Bohužel jej není možné použít pro zakončení VPN tunelů, nemusí umožňovat překlad síťových adres nebo jeho nastavení či řešení problémů nemusí být tak přímočaré, jako v případě firewallu netransparentního [2]. V této práci se primárně zaměříme na tento druh firewallu.



Obrázek 2.2: Schéma funkčnosti transparentního firewallu.

2.2. Obvyklé umístění firewallů v síti

Z předchozích odstavců není úplně těžké odhadnout, jaké druhy firewallů se v kterých částech sítě používají. Základním vybavením každé koncové stanice by měl být osobní firewall, který určí, jaký provoz může stanice přijímat a odesílat. Pokud stanice nemůže být vybavena osobním firewallem, možností je umístit před tuto stanici transparentní firewall, který se o filtraci síťového provozu postará. Díky transparentnosti není ani potřeba řešit předadresování stanice. Na perimetru chráněné sítě by se měl nacházet síťový firewall. Jedná se většinou o (stavový) packetový filtr netransparentního typu. Za něj se umisťuje aplikační brána (spíše ale více bran, pro každý analyzovaný protokol zvlášť). Tato topologie je výhodná v tom, že packetový filtr provede rychlé hrubé odfiltrování nežádoucího provozu a zbytek je důkladněji prověřen v aplikačních bránách [1]. Kromě netransparentního packetového filtru je možné použít kombinaci transparentního packetového filtru s klasickým směrovačem, kdy se transparentní firewall umístí do vnější sítě před směrovač. Tím je možné odfiltrovat provoz ještě předtím, než dorazí na hraniční směrovač.

3. Vysokorychlostní zpracování a filtrace packetů v Linuxu

3.1. Tradiční přístup ke zpracování packetů

Jak již bylo naznačeno v úvodu, Linux je univerzální jádro operačního systému, které v oblasti síťového zásobníku nabízí bohaté možnosti, jež se hodí pro různé aplikace. Bohužel tento přístup s sebou přináší limity v oblasti výkonu. Hlavním problémem nejen Linuxu, ale i dalších univerzálních operačních systémů, je výkon zpracování vstupně-výstupních (IO) operací. Obvyklý způsob jejich zpracování je pomocí přerušení (IRQ). Přerušení slouží k upozornění procesoru, že například síťová karta přijala nový packet a je nutné jej zpracovat. Díky tomuto způsobu zpracování IO operací se může procesor v čase mezi přerušeními věnovat jiné činnosti. Přerušení se však hodí pro prostředí, kdy systém není příliš zatížen IO operacemi. V případě vysokého síťového zatížení, s čímž se dá v případě firewallu počítat, se přerušení stávají značně neefektivní metodou zpracování vstupně-výstupních operací, jelikož každý IRQ je drahou operací. Snadno se tak může stát, že procesor začne být zatížen jen vyřizováním požadavků na přerušení, místo toho, aby zpracovával samotné packety. Tento jev se označuje jako „interrupt livelock“ [3].

Druhou možností, jak ke zpracování vstupně-výstupních operací přistoupit, je tzv. „polling“. Při této metodě se procesor aktivně a pravidelně vyptává síťové karty, zda přijala nová data, která je potřeba vyřídit. Jelikož je polling méně nákladnou operací než přerušení, lze tím docílit vyšší propustnosti a snížení latencí, ovšem za cenu alokování procesoru jen pro tuto činnost.

Proto byl v Linuxu verze 2.4.20 představen hybridní přístup pojmenovaný NAPI (New API) [4]. NAPI funguje tak, že po startu systému je nastaven režim přerušení. Při přijetí nového packetu je vyvolán požadavek na přerušení. Poté jsou pro dané zařízení přerušení zakázána a přechází se do režimu pollingu. V tomto režimu procesor pravidelně přistupuje k síťové kartě a dává pokyn k přesunu dat z vyrovnávací paměti síťové karty do paměti RAM. Dokud jsou ve vyrovnávací paměti data, je dané zařízení plánováno stále v režimu pollingu. V opačném případě se polling zakáže a opět se povolí přerušení. Tímto přístupem lze dosáhnout propustnosti až několika Gbit/s [5], ovšem ovladač síťové karty musí NAPI podporovat.

3.2. Možnosti zrychlení zpracování packetů

V předchozí sekci bylo vysvětleno, proč Linux není efektivní při vysoké síťové zátěži a jakým způsobem se to samotné jádro snaží řešit. Ovšem existují i jiné způsoby, jakými lze dosáhnout vyšší datové propustnosti [3]:

- **Obejít síťový zásobník jádra** – jednou z nejúčinnějších metod, jak zrychlit zpracování packetů, je vůbec je nevpouštět do jádra a jeho síťového zásobníku, ale zpracovávat je v uživatelském prostoru (user-space) pomocí vlastního síťového zásobníku, který bude přesně odpovídat požadavkům aplikace. Tímto způsobem lze dosáhnout vysoké efektivity zpracování, ovšem je třeba upravit ovladač síťové karty tak, aby

nepředával data jádru, ale uživatelské aplikaci. Jádro tedy neuvidí žádná síťová data, a tudíž není možné využít jakýchkoliv jiných jeho síťových služeb.

- **Využívat polling místo přerušení** – pro systémy s vysokým síťovým zatížením je efektivnější využívat polling místo drahých přerušení, jak bylo vysvětleno v 3.1.
- **Kopírovat data co nejméně** – je logické, že čím více se budou zpracovávaná data kopírovat (např. mezi síťovým rozhraním a jádrem, jádrem a uživatelskou aplikací), tím větší bude zpoždění celkového zpracování. Ideálně by měl ovladač síťového rozhraní zkopírovat data do paměti a přímo s touto pamětí by měla aplikace pracovat. Tímto lze dosáhnout i velice rychlého předávání dat mezi rozhraními – stačí v paměti data označit jako odchozí z jiného rozhraní.
- **Zpracovávat data v dávkách** – místo provádění operací nad každým packetem zvlášť, je efektivnější je zpracovávat v dávkách několika packetů. Tím se náklady spojené s danou operací rozloží do více packetů.
- **Alokace paměti předem** – pro každou alokaci paměti je potřeba vyvolat systémové volání, při kterém musí jádro nalézt dostatečně velký paměťový prostor. Každé takové volání samozřejmě stojí čas. Proto se jako vhodná technika jeví alokace paměti třeba při startu aplikace. Jelikož se velikost paměti při běhu aplikace již nemůže měnit, je třeba zvolit takovou velikost, která bude dostatečná po celou dobu běhu aplikace.

3.3. Nástroje a knihovny pro vysokorychlostní zpracování a filtraci

3.3.1. netfilter + iptables

Dobře známý filtrovací podsystém Linuxového jádra, jehož je standardní součástí od verze 2.4. Je velice univerzální, a proto jej lze použít v mnoha různých situacích: od prostého packetového filtru přes stavový packetový filtr až po zařízení provádějící překlad síťových adres či jinou změnu v hlavičkách packetů. Může pracovat jako osobní, ale i síťový, firewall. Netfilter vytváří v jádře několik záchytných bodů (hooks) v různých fázích zpracování packetu. Dynamicky načítané jaderné moduly pak mohou k těmto bodům zaregistrovat svou callback funkci a tím zajistit, že bude-li packet procházet daným bodem, bude předán k prozkoumání i zaregistrovanému modulu. Počet registrací k danému záchytnému bodu není omezen a packet je postupně předán všem registrovaným modulům (pokud není některým zahozen) [6]. Nejznámější utilitou pro správu pravidel a jaderných modulů je zřejmě `iptables`. Kromě `iptables` ale existují i jiné utility, např. `ip6tables` nebo `ebtables`. Právě druhá zmíněná utilita slouží k filtrování na úrovni linkové vrstvy ISO OSI modelu. Pomocí `ebtables` lze filtrovat až do transportní vrstvy, ovšem pouze podle adres (IP nebo portů) a bezstavově. Díky modulu `bridge-netfilter` (označovaný i jako `br-netfilter` nebo `bridge-nf`) lze i při transparentním filtrování využívat možností `iptables`, tzn. provádět například stavovou inspekci či překlad síťových adres. V jádrech 3.18-rc1 a novějších je potřeba tento modul načíst, ve starších jádrech je integrován přímo v modulech `iptables`, takže jej není nutné explicitně načítat.

3.3.2. nftables

Nový podsystém pro filtrování a úpravu síťového provozu v Linuxu. Součástí jádra od verze 3.13. Využívá infrastrukturu Netfilteru (záchytné body), ovšem dynamicky načítané moduly jsou jiné než využívají `iptables` a další. Moduly v jádře vytváří virtuální stroj, který pracuje se speciálním byte kódem, do kterého jsou zkompilovány jednotlivá filtrovací pravidla. Nabízí přímou možnost pro vkládání množin parametrů (např. IP adres nebo portů) do jednoho pravidla, takže je možné sloučit více pravidel, a tím zrychlit celkové vykonávání filtrace. O uživatelské rozhraní se stará nová utilita `nft`, která sjednocuje `iptables` a další `*tables` utility. Syntaxe pravidel je zcela odlišná od syntaxe netfilter utilit. `nft` slouží zároveň jako kompilátor a dekompilátor pro byte kód, který nahrává do/z jaderného virtuálního stroje. Tím je přesunuta velká část logiky do uživatelské utility a jádro tak může pracovat jen s byte kódem [7].

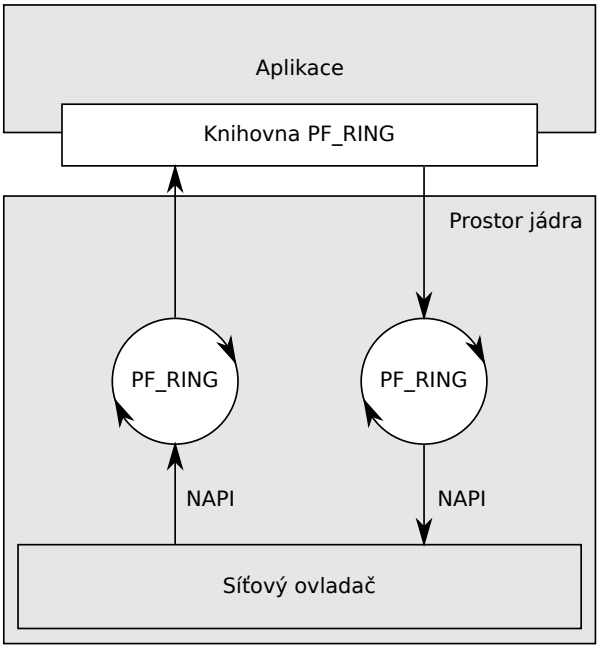
3.3.3. PF_RING

PF_RING je knihovna, resp. sada knihoven (framework), a jaderný modul, pomocí nichž lze zpracovávat síťový provoz při vysoké rychlosti. Využívá principů popsaných v sekci 3.2. PF_RING dokáže pracovat jak s běžnými ovladači síťových karet, tak nabízí upravené ovladače karet Intel a dalších výrobců, pomocí kterých lze linuxové jádro zcela obejít a dosáhnout tak vyšší datové propustnosti. Bohužel jsou tyto ovladače použitelné pouze po zakoupení licence [9]. Při využití standardních ovladačů jsou v jádře vytvořeny dvě cyklické fronty (jedna pro příjem, druhá pro vysílání), které s ovladačem karty komunikují pomocí NAPI. Z těchto front pak čtou (příp. do nich zapisují) uživatelské aplikace pomocí PF_RING knihovny (viz obrázek 3.1).

PF_RING nabízí API pro filtrování provozu pomocí syntaxe BPF (stejně jako `tcpdump` například), ale dokáže využít i možnosti filtrovat provoz hardwarově přímo v síťové kartě. Hardwarový filtr je však podporován pouze minimálním počtem karet, proto mu dále nebude věnována pozornost. Ačkoliv zřejmě neexistuje žádná volně dostupná implementace firewallu využívající PF_RING, v rámci jeho zdrojových kódů je distribuována i příkladová aplikace `pfbridge`, která umožňuje přemostit dvě síťová rozhraní a provádět filtraci průchozího provozu. Využívá právě zmíněnou syntaxi BPF a samotný filtr označuje provoz, který má být propuštěn (tudíž jen ty packety, které vyhoví filtru, budou propuštěny, vše ostatní bude zahozeno).

3.3.4. netmap

Stejně jako PF_RING se i netmap skládá z knihoven a jaderného modulu. Původně byl vyvíjen pro FreeBSD, ovšem nyní existují verze pro Linux, a dokonce i Windows. Svou architekturou je velice podobný PF_RINGu – cyklické fronty pro každý směr a každé síťové rozhraní, do kterých zapisují a ze kterých čtou jak ovladače síťových karet, tak aplikace. Pro vysokou rychlost zpracování jsou vyžadovány upravené ovladače karet, ale netmap nabízí i emulační mezivrstvu, s jejíž pomocí je možné používat standardní jaderné ovladače. V případě, že není aktivní žádná netmap aplikace, chovají se ovladače obvyklým způsobem (předávají data jadernému síťovému zásobníku). Po spuštění netmap aplikace je ovladač přepnut do tzv. „netmap módu“, během kterého nedostává jádro žádná data, ale všechna jsou předávána netmap aplikaci. Po ukončení této aplikace je opět přepnuto do původního režimu [5].



Obrázek 3.1: Schéma funkčnosti knihoven a jaderného modulu PF_RING [8].

Pro demonstraci možností a dobré kompatibility API, implementoval autor netmapu FreeBSD firewall `ipfw` pomocí frameworku netmap. Díky tomu je možné celkem snadno zprovoznit vysokorychlostní packetový filtry umožňující i tvarování provozu na běžně dostupném hardwaru.

3.3.5. Data Plane Development Kit

Data Plane Development Kit neboli DPDK je další skupina knihoven, jenž umožňuje zpracování síťového provozu při vysoké rychlosti s minimálním zatížením procesoru. Opět staví na principech ze sekce 3.2 a sdílených cyklických frontách. Kromě samotné akcelerace zpracování síťových dat nabízí i jiné funkce, které se mohou hodit při vývoji komplexních síťových aplikací, např. algoritmy pro hledání nejdelšího prefixu využívané při implementaci L3 směrování či QoS funkce. DPDK je podporován mnoha výrobci, proto nabízí zřejmě nejvíce upravených ovladačů síťových karet ze všech zmíněných frameworků [10]. Nenabízí podporu pro žádný transparentní nebo hybridní mód, je-li tedy síťová karta ovládána pomocí DPDK, nejsou žádná data předávána jádru, ať už běží nějaká DPDK aplikace nebo ne. Veškeré části DPDK včetně ovladačů fungují pouze na principu pollingu a nevyužívají tedy žádným způsobem přerušení. Ovladače síťových karet provádí značnou část svých operací v uživatelském prostoru, na jádro je ponechána pouze inicializace karty. Tento speciální typ ovladače se označuje UIO [3].

Ačkoliv je DPDK hojně podporovaný výrobci, a někteří výrobci staví své produkty umožňující filtraci packetů nad tímto frameworkem (např. [11] nebo [12]), volně dostupných filtrovacích nástrojů příliš neexistuje. Pro názornou ukázku možností DPDK jsou v rámci jeho zdrojových kódů šířeny DPDK aplikace `l3fwd-acl` a `ip_pipeline`, které slouží k ukázce bezstavového (ne)transparentního packetového filtru.

3.3.6. Snabb

Snabb, dříve známý jako SnabbSwitch, je síťový framework kompletně napsaný v odlehčeném vysokoúrovňovém skriptovacím jazyce Lua. Aby byl omezen dopad nižší výkonnosti skriptovacích jazyků oproti kompilovaným jazykům, je využívána JIT kompilace LuaJIT. Díky zvolenému jazyku jsou před programátory schovány implementační detaily, a proto je možné psát vysokorychlostní síťové aplikace snadno a pomocí malého počtu řádků kódu. Snabb nevyužívá jaderné ovladače síťových karet, ani nějaké jejich upravené verze, ale dodává své vlastní ovladače, taktéž napsané v jazyce Lua a využívající přístupu UIO. Kvůli této závislosti jsou podporovány prakticky jen karty Intel.

Snabb API nabízí mnoho různých funkcí, včetně podpory pro filtrování pomocí BPF. Bohužel žádná z příkladových aplikací tuto funkčnost prakticky nedemonstruje. O implementaci kompletního firewallu nad frameworkem Snabb se snaží projekt SnabbWall, který v době psaní této práce měl zatím implementovánu část detekcí toků, ale samotný packetový filtr ještě nebyl vyvinut [13].

3.3.7. PFQ

PFQ je síťový framework napsaný částečně v jazyce C/C++ a částečně ve funkcionálním jazyce Haskell. Je optimalizovaný pro vícejádrové procesorové architektury a síťové karty s více hardwarovými frontami. Nepotřebuje ke své funkci upravené ovladače a zaměřuje se spíše na zpracování packetů v prostoru jádra než v uživatelském prostoru. Do linuxového jádra je načten v podobě modulu, se kterým pak komunikují uživatelské aplikace přes knihovny, které jsou dostupné pro jazyky C, C++ a Haskell. Samotný modul využívá pro komunikaci s ovladačem síťové karty rozhraní NAPI.

Jako některé další zmíněné frameworky, nabízí PFQ ve svém API funkce pro filtrování packetů pomocí BPF či VLAN hlavičky. Bohužel aplikaci demonstrující tuto funkci zřejmě není možné najít.

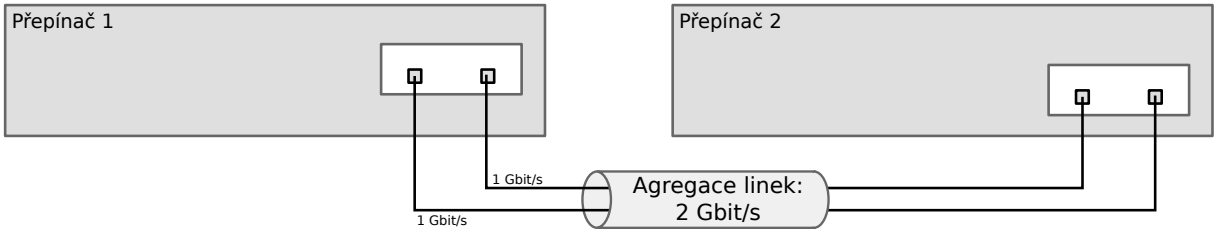
3.3.8. nf-HiPAC

nf-HiPAC je framework pro vysokovýkonnostní klasifikaci packetů pro netfilter. Pro optimalizaci výkonu využívá algoritmy, které snižují počet přístupů do paměti. Zachovává syntaxi používanou utilitou `iptables`, avšak podporuje pouze filtraci packetů (ať už stavovou či bezstavovou). Překlad síťových adres či jiná změna hlaviček packetů není podporována [14].

I když vypadá tento projekt slibně, je již více než 10 let neaktivní a podporuje pouze jádra 2.6.11 až 2.6.14 [15]. Dnes již tyto verze nejsou podporovány, a z bezpečnostního pohledu o nich nemá cenu uvažovat.

4. Linková agregace

Linkovou agregací se myslí technika, při které se více fyzických propojení mezi dvěma síťovými prvky chová stejně jako jedno logické propojení nabízející vyšší datovou propustnost (danou součtem rychlostí jednotlivých fyzických rozhraní, viz obrázek 4.1) a zvýšení dostupnosti propojení mezi danými prvky [16]. V případě výpadku jednoho z fyzických propojů, je možné i nadále přenášet data zbývajícími propojeními bez zaznamenání daného selhání, pouze se sníží přenosová kapacita propojení. Díky existenci standardu IEEE 802.3ad nebo nověji IEEE 802.1AX-2008, je možné agregovat linky mezi zařízeními různých výrobců. Standardy kromě statické agregace, kdy je nutné agregaci ručně nastavit na obou zařízeních, definuje i LACP, což je L2 protokol pro dynamickou konfiguraci linek pro agregaci. Při agregaci musí mít všechny linky stejné parametry (linkovou rychlost, duplex atd.).



Obrázek 4.1: Příklad, jakým způsobem funguje agregace linek.

Na Linuxu existují pro agregaci linek dvě hlavní řešení – NIC bonding a NIC teaming. Kromě některých vylepšení, které NIC teaming nabízí oproti bondingu [17], je hlavní rozdíl mezi nimi v umístění samotné logiky – zatímco při bondingu jsou veškeré operace prováděny v prostoru jádra pomocí zásuvného modulu, NIC teaming funguje jako démon v uživatelském prostoru. Obě tato řešení jsou však kompatibilní jen se standardním síťovým zásobníkem Linuxu. V případě použití filtrovacích nástrojů, které obchází linuxový zásobník, je třeba linkovou agregaci řešit ve spolupráci s těmito nástroji. Jediným frameworkem, u kterého byl nalezen popis agregace linek, je DPDK [18].

Aby mohla být filtrace při agregovaných linkách vyzkoušena, je potřeba nejen podpora v samotných firewallech, ale také celé síťové infrastruktury. Jak je popsáno dále v textu, při testování v experimentální síti se podařilo infrastrukturu přepojit tak, aby mohla být linková agregace a filtrace vyzkoušena, ovšem jen pro dvě 1Gbit/s (tedy do maximální přenosové rychlosti 2 Gbit/s).

5. Praktické testování vysokorychlostní filtrace v Linuxu

Praktické testování nástrojů a knihoven pro vysokorychlostní filtraci síťového provozu na běžném hardwaru bylo rozděleno do několika částí. Aby nebylo nutné alokovat drahé prostředky (hlavně přepínač a porty na generátoru provozu) pro 10Gbit/s rychlost na dlouhou dobu, byly v první fázi nastaveny a otestovány filtrovací nástroje na rychlosti do 1 Gbit/s a až poté, co bylo testovací prostředí nastaveno, se síť přepojila na plně 10 Gbit/s přenosovou rychlost.

Po odzkoušení filtrace při rychlosti dat 10 Gbit/s bylo testování filtračních nástrojů přesunuto na experimentální pracoviště, kde byly nástroje testovány při různých nastaveních filtrace, a také byla vyzkoušena možnost filtrace na agregovaných linkách.

5.1. Výběr nástrojů a knihoven pro testování

Již ze shrnutí v sekci 3.3 lze částečně odvodit, které nástroje a knihovny nabízí ukázkové či samostatně vyvíjené programy umožňující filtraci síťového provozu a které ne, či se k testování v rámci této práce nehodí. Zcela určitě by ve výběru neměla chybět v dnešní době klasická varianta filtrování pomocí netfilteru a `iptables`, aby se potvrdilo či vyvrátilo, že se linuxové jádro pro vysokorychlostní filtraci dat opravdu nehodí. Dále byl vybrán k otestování nástroj nftables, aby se mohlo případně projevit, zda s novým filtrovacím podsystémem přichází i vyšší výkonnost filtrace. Mezi další vybrané patří framework PF_RING. Ten v rámci svých zdrojových kódů distribuuje ukázkovou aplikaci, která umožňuje přemostit dvě síťová rozhraní a aplikovat při tom určitý filtr. Jako poslední byl vybrán framework netmap, pomocí nějž byl implementován firewall `ipfw` označovaný jako `netmap-ipfw`. Ostatní nástroje a knihovny buď nenabízí již implementované filtrovací programy, anebo se pro tuto práci nehodí. Frameworky Snabb a PFQ v době psaní této práce nenabízely nástroje pro filtrování. nf-HiPAC se zase nehodí z důvodu požadavku provozování na velice starém jádře. DPDK sice nabízí ukázkové filtrovací aplikace, ovšem buď se nejedná o transparentní filtraci (`ip_pipeline`) anebo umožňuje filtrovat transparentně (`l3fwd-acl`), ale poradí si jen s protokoly na síťové a transportní vrstvě (IP, TCP, UDP), ale ne na linkové vrstvě (ARP), což vytváří požadavek na statické definování záznamů v ARP tabulce. Tento problém se projevil při počátečním seznamování se s frameworkem a na jeho základě bylo DPDK z výběru pro testování vyloučeno.

Výsledný seznam tedy je: netfilter + `iptables`, nftables, PF_RING a netmap (respektive firewall `netmap-ipfw`). Verze a licence nástrojů ve výběru lze nalézt v tabulce 5.1.

Název	Licence	Verze
netfilter + `iptables`	GPLv2	1.4.21 (Linux 3.16)
nftables	GPLv2	v0.6 (Linux 4.7, viz dále 5.4.4)
PF_RING	LGPL 2.1	vývojová verze, listopad 2016
netmap + `netmap-ipfw`	BSD 2-Clause	vývojová verze, listopad 2016

Tabulka 5.1: Verze a licence vybraných nástrojů pro otestování v první a druhé fázi testování.

5.2. Testovací prostředí

5.2.1. Generátor provozu – Spirent Avalanche 3100B

Spirent Avalanche 3100B je síťové hardwarové zařízení, které slouží k testování jiných zařízení a aplikací. Dokáže testovat jak výkon daného zařízení/aplikace, tak odzkoušet zabezpečení či kvalitu dostupných služeb. Pracuje na transportní až aplikační vrstvě referenčního modelu ISO OSI. Zařízení, které bylo použito k vypracování této práce, má k dispozici 2× 10Gbit/s SFP+ a 12× 1Gbit/s RJ-45 Ethernet rozhraní, dokáže vygenerovat více něž jeden milion HTTP GET dotazů a vytvořit jimi datový tok několika desítek Gbit/s. Obecně dokáže vytvořit a udržovat až 30 milionů stavových spojení. Všechny důležité informace a podporované protokoly lze najít v datasheetu [19]. Zátěž lze v zařízení Spirent Avalanche 3100B definovat různými způsoby, např. určením požadované datové propustnosti či nastavením počtu simulovaných uživatelů. Pro náš případ se však osvědčilo nastavení počtu provedených transakcí za sekundu. Transakcí je zde myšleno například stažení jedné webové stránky nebo jeden DNS dotaz.

5.2.2. Testovací server – IBM System x3550 M2

Jako testovací stroj byl zvolen server značky IBM a typu System x3550 M2. Server obsahuje dva osmijádrové procesory Intel Xeon L5520 pracujících na frekvenci 2,27 GHz. Operační paměť má kapacitu 48 GB a pevný disk dokáže uložit až 350 GB dat. Na základní desce se také nachází dvě síťová rozhraní Broadcom NetXtreme II BCM5709 Gigabit Ethernet. První z těchto rozhraní bylo použito pro vzdálenou správu systému. Dále byla do serveru přidána PCI Express síťová karta Intel 10-Gigabit X540-AT2, jež byla použita pro testování nástrojů a knihoven pro filtraci síťového provozu. Karta Intel obsahuje dvě metalická Ethernet rozhraní, která podporují linkové rychlosti 100 Mbit/s, 1 Gbit/s a 10 Gbit/s, ale pouze ve full-duplexním režimu. Linuxový ovladač pro tato rozhraní se načítá jako jaderný modul a má označení `ixgbe`. Jako operační systém byla zvolena linuxová distribuce Debian ve stabilní verzi 8 (kódové označení Jessie) s jádrem Linux 3.16.

5.3. Testovací metody

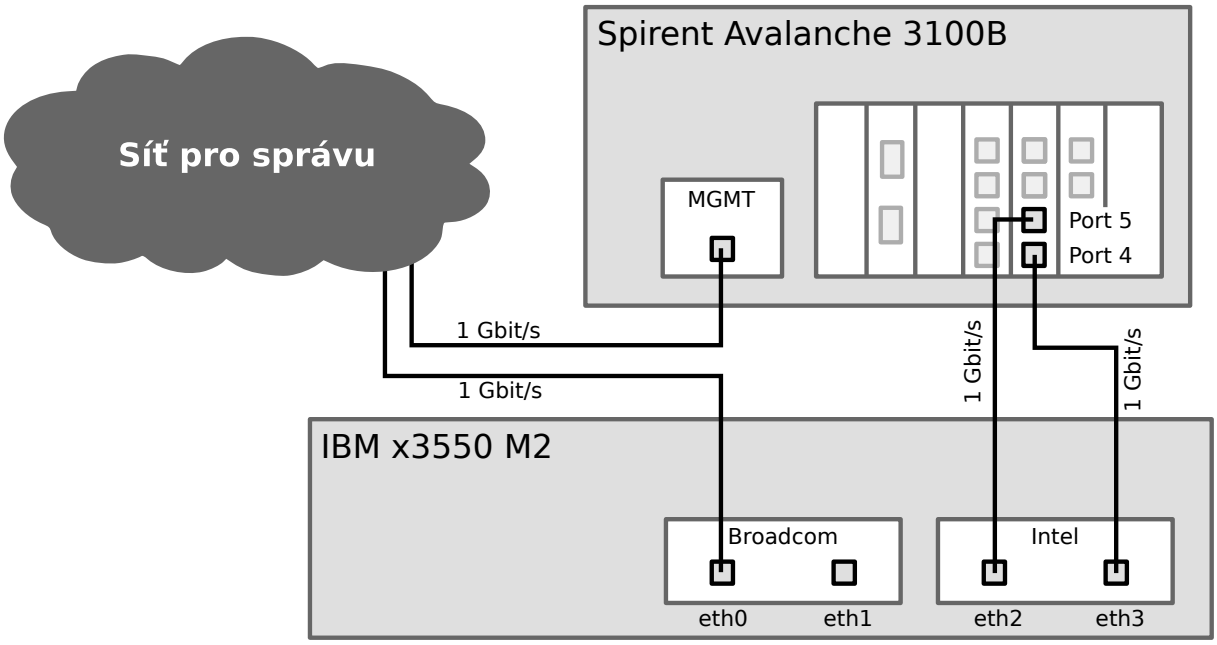
Pro otestování funkčnosti a výkonnosti jednotlivých filtrovacích nástrojů byl generátor provozu nastaven tak, aby generoval provoz pro tři různé protokoly – TCP, UDP a ICMP. Z protokolů nad TCP byl vybrán protokol HTTP, který komunikuje na serveru na portu 80, z aplikačních protokolů nad UDP byl zvolen protokol DNS, který komunikuje na serveru na portu 53, a z ICMP byl vybrán typ zprávy 8 – „Echo request“, který je generován známým nástrojem `ping`. Při zkoušení jednotlivých filtrovacích nástrojů bude tedy test opakován několikrát, kdy v každém opakování bude zahazován jeden z protokolů plus bude proveden jeden test na vyzkoušení filtrace podle zdrojové IP adresy. Jelikož Spirent Avalanche 3100B dokáže generovat provoz z více zdrojových adresních rozsahů, které pak ve výsledcích dokáže od sebe oddělit, v případě filtrování podle zdrojové adresy nás bude zajímat, zda z jednoho rozsahu prošel veškerý provoz a z druhého žádný. V případě filtrování podle protokolů a portů nás bude zajímat, zda byl vybraný protokol zablokován

a ostatní povoleny. Jinými slovy – ze zablokovaných adresních rozsahů či protokolů musí být všechny transakce neúspěšné, ostatní transakce by měly být úspěšné.

5.4. První fáze – testování do rychlosti 1 Gbit/s

5.4.1. Síťová topologie

Síťová topologie testovacího prostředí byla zvolena tak, aby žádné další zařízení nemohlo zasahovat do síťového provozu a jakýmkoliv způsobem jej ovlivňovat. Proto byl generátor provozu připojen přímo k testovacímu serveru pomocí metalické UTP kabeláže kategorie 6. Jelikož testovací server slouží pouze k transparentnímu filtrování, provoz generovaný zařízením Spirent Avalanche 3100B musel být někde zakončen a zpracován. Pro tento účel byl opět využit již zmíněný Spirent Avalanche 3100B, který může zároveň fungovat jako generátor (klientská část), tak i příjemce (serverová část). Proto byl testovací server propojen s generátorem (a příjemcem) provozu dvěma rozhraními (viz obrázek 5.1). V tomto případě sloužil port 4 pro generování provozu a port 5 pro zpracování provozu. Vezmeme-li v potaz informace z odstavce 5.2.1, je očividné, že maximální linková rychlost tohoto zapojení je 1 Gbit/s. Je to z toho důvodu, že metalická rozhraní na zařízení Spirent Avalanche 3100B jsou „jen“ gigabitové. To nám ovšem pro počáteční seznámení se s filtrovacími nástroji a jejich vyzkoušení nevadí. Testovací server i generátor Avalanche 3100B byl pak také jedním rozhraním připojen do sítě pro vzdálenou parametrizaci a správu.



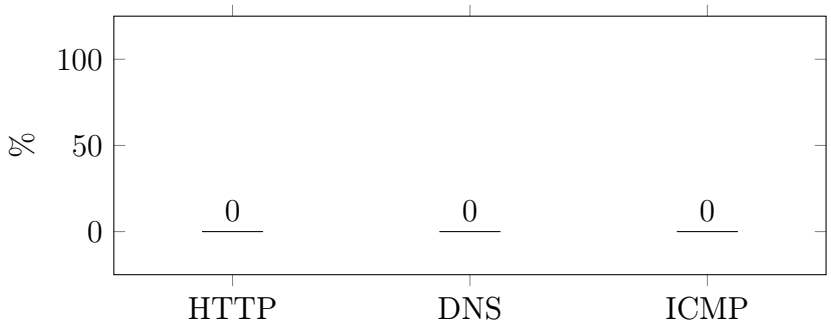
Obrázek 5.1: Tímto způsobem byla zařízení v první fázi síťově propojena.

5.4.2. Nastavení Spirent Avalanche 3100B

Jak bylo popsáno dříve, Spirent Avalanche 3100B funguje jako generátor provozu, tak i jeho příjemce. Proto je nutné nastavit jak klientkou část, tak i serverovou část. Pro server byla zvolena IP adresa `192.168.1.1`, na níž bude směrován generovaný provoz. Zdrojové IP adresy klientů byly rozděleny do dvou rozsahů – první (dále označovaný jako rozsah1) obsahuje adresy `192.168.1.10-192.168.1.100`, druhý (dále označovaný jako rozsah2) obsahuje adresy `192.168.1.110-192.168.1.200`.
Samotné nastavení testu bylo provedeno tímto způsobem:

- zátěž byla nastavena na 22 000 transakcí za sekundu,
- test probíhal 25 sekund a byl nastaven tak, aby ihned po startu začal generovat nastavenou zátěž,
- akce, které probíhaly během testu, byly následující:
  - `get http://192.168.1.1/index.html` – jedná se o HTTP GET, který vyžaduje stažení 6kB webové stránky nacházející se na daném URL,
  - `ICMP://192.168.1.1 ECHO LENGTH=64` – odeslání 64 B dlouhého ICMP Echo požadavku na adresu `192.168.1.1`,
  - `DNS A 192.168.1.1 www.spirent.com` – odeslání DNS dotazu na přeložení domény www.spirent.com na DNS server `192.168.1.1`,
- tímto nastavením byl vygenerován provoz o průměrné velikosti cca 618 Mbit/s (referenční měření, viz dále).

Ještě před samotným testováním nástrojů bylo provedeno referenční měření s výše uvedeným nastavením, kdy bylo klientské rozhraní (port 4) připojeno kabelem přímo na na rozhraní serverové (port 5). Výsledek referenčního měření shrnuje tabulka A.1 a graf 5.1.



Graf 5.1: Procentuální vyjádření neúspěšných transakcí při referenčním měření podle protokolu.

5.4.3. netfilter + iptables

Instalace a nastavení

V tomto případě není potřeba cokoliv instalovat. V Debianu, ale i dalších linuxových distribucích, je netfilter standardní součástí jádra a `iptables` bývá výchozím firewallem,

takže ihned po instalaci bývá k dispozici. Co už ovšem je potřeba nastavit, je přemostění síťových rozhraní, na kterém se pak bude provádět filtrace. To lze provést například tímto způsobem:

```
1   # ip link add name br0 type bridge
2   # ip link set br0 up
3   # ip link set eth2 up master br0
4   # ip link set eth3 up master br0
```

Znak `#` na začátku řádku znamená, že je tento příkaz vykonáván uživatelem `root`.

První řádek vytváří nové rozhraní `br0` typu most. V druhém řádku se nové rozhraní spustí. Řádky 3 a 4 zařadí fyzická rozhraní `eth2` a `eth3` (viz obrázek 5.1) do nově vytvořeného mostu, a také je spustí. Tím se začnou fyzická rozhraní chovat jako rozhraní přepínače. Nyní je možné spustit testy a testovat filtraci.

Jelikož je ve stabilní verzi Debianu verze jádra 3.16, není nutné načítat žádný jaderný modul (viz 3.3.1). V případě novějších jader by bylo potřeba načíst zásuvný modul `bridge-netfilter`, což by se provedlo následujícím příkazem:

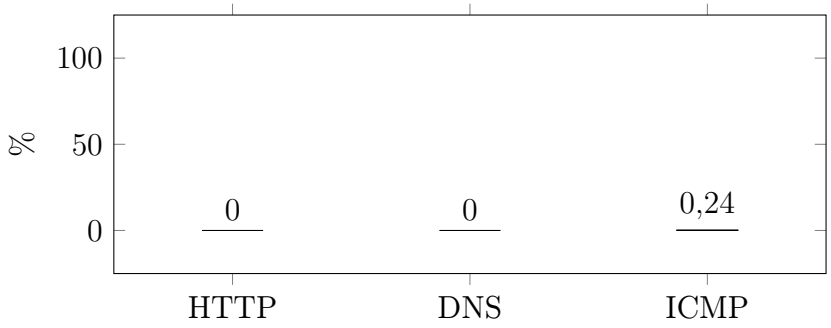
```
1   # modprobe br_netfilter
```

Testování – bez filtrace

Nejdříve provedeme spuštění testu bez samotné filtrace. Filtrovací tabulka `iptables` je tedy prázdná:

```
1   Chain INPUT (policy ACCEPT)
2   target     prot opt source               destination
3
4   Chain FORWARD (policy ACCEPT)
5   target     prot opt source               destination
6
7   Chain OUTPUT (policy ACCEPT)
8   target     prot opt source               destination
```

Z tabulky A.2 a grafu 5.2 vyplývá, že došlo k nechtěnému zahození 679 ICMP zpráv (tj. 0,24 % ICMP provozu). Mohlo by to poukazovat na určité limity přemostění síťových rozhraní v Linuxu. Během měření bylo dosaženo průměrné datové propustnosti zhruba 616 Mbit/s.



Graf 5.2: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – bez filtrace.

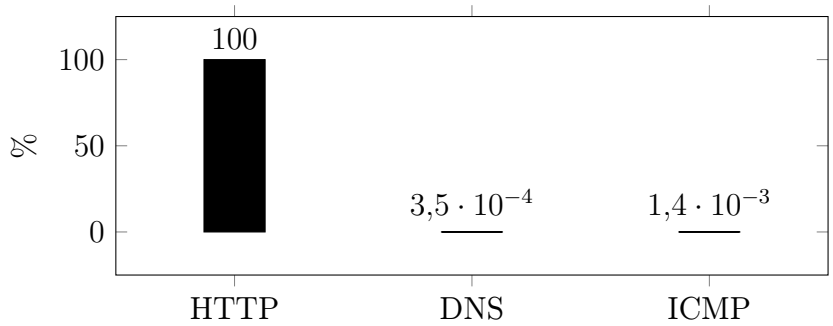
Testování – filtrace HTTP

Pro filtrování HTTP zadáme tento příkaz:

```
1   # iptables -A FORWARD -m physdev --physdev-is-bridged \
2         -p tcp -m tcp --dport 80 -j DROP
```

Příkaz přidá do řetězce `FORWARD` filtrovací tabulky pravidlo, které se aplikuje pouze na provoz, který je jádrem přemostěn (část `-m physdev --physdev-is-bridged`) a jehož transportní protokol je TCP a cílový port 80. Pokud nějaký packet odpovídá těmto podmínkám, bude zahozen.

Výsledek měření shrnuje graf 5.3 a tabulka A.3. Jak je vidět, veškerý HTTP provoz byl opravdu odfiltrován a neproběhla žádná úspěšná transakce na tomto protokolu. Kromě toho došlo k nechtěnému zahození 4 ICMP zpráv a jedné DNS transakce, což je ale zanedbatelné množství.



Graf 5.3: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace HTTP.

Testování – filtrace DNS

Pro filtraci DNS zadáme podobný příkaz jako u HTTP, ovšem s jiným portem a jiným transportním protokolem:

```
1   # iptables -A FORWARD -m physdev --physdev-is-bridged \
2         -p udp -m udp --dport 53 -j DROP
```

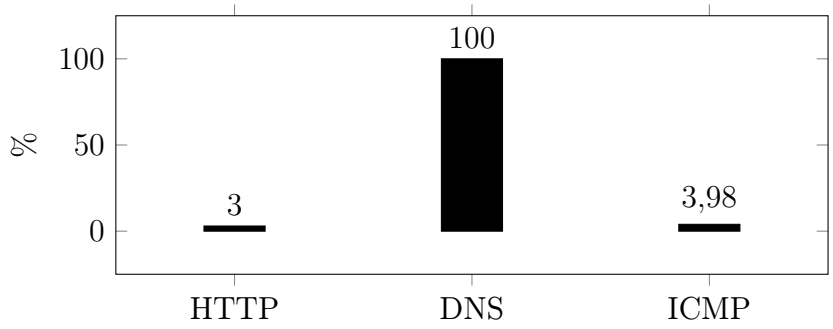
Příkaz je velice podobný příkazu pro filtrování HTTP provozu, proto zřejmě nepotřebuje další vysvětlování.

Jak je z tabulky A.4 vidět, DNS bylo správně zcela odfiltrováno. Jak ovšem ukazuje graf 5.4, došlo i k velké nechtěné filtraci HTTP a ICMP. Není zcela jasné, čím byla tato situace způsobena.

Testování – filtrace ICMP

Velice podobně jako v předchozích případech zapíšeme pravidlo pro filtraci ICMP:

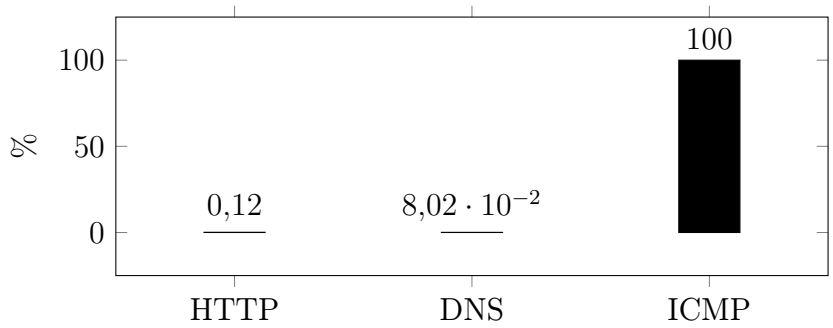
```
1   # iptables -A FORWARD -m physdev --physdev-is-bridged \
2         -p icmp -m icmp --icmp-type 8 -j DROP
```



Graf 5.4: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace DNS.

Jelikož není ICMP transportní protokol, nefiltrujeme zde podle čísla portu, ale podle typu ICMP zprávy.

V grafu 5.5 můžeme vidět, že ICMP bylo opravdu odfiltrováno, ovšem spolu s ním částečně i ostatní protokoly (viz také tabulku A.5).



Graf 5.5: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace ICMP.

Testování – filtrace rozsahu IP adres

V případě, že chceme filtrovat rozsah IP adres, máme několik možností, jak to udělat. Buď můžeme zadat zdrojovou adresu jako podsíť (adresa sítě + prefix), nebo pokud nechceme celou podsíť, ale jen její část, můžeme využít rozšíření `iprange`:

```
1   # iptables -A FORWARD -m physdev --physdev-is-bridged -m iprange \
2         --src-range 192.168.1.110-192.168.1.200 -j DROP
```

Jak je v příkazu vidět, používáme rozšíření `iprange` a argumentem `--src-range` definujeme rozsah zdrojových adres, kterých se má pravidlo týkat.

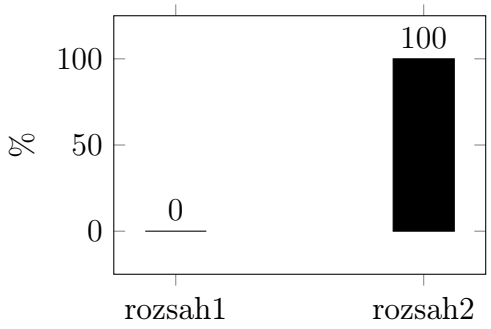
Tabulka A.6 a graf 5.6 dokazují, že opravdu došlo k odfiltrování provozu z vybraného rozsahu IP adres, kdežto druhý rozsah zůstal povolen.

5.4.4. nftables

Instalace a nastavení

Od verze jádra Linux 3.13 jsou nftables jeho standardní součástí, proto je potřeba doinstalovat jen utilitu `nft`. To v prostředí Debianu a jeho derivátů provedeme příkazem:

```
1   # apt-get install nftables
```



Graf 5.6: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace rozsahu IP adres.

Tím dojde k nainstalování ovládací aplikace. Na rozdíl od `iptables` nemají nftables přednastavené tabulky a řetězce pro přidávání pravidel. K tomuto účelu jsou v adresáři `/usr/share/doc/nftables/examples/syntax/` ukázkové skripty, které lze využít. My využijeme soubor `bridge-filter`:

```
1   # nft -f /usr/share/doc/nftables/examples/syntax/bridge-filter
```

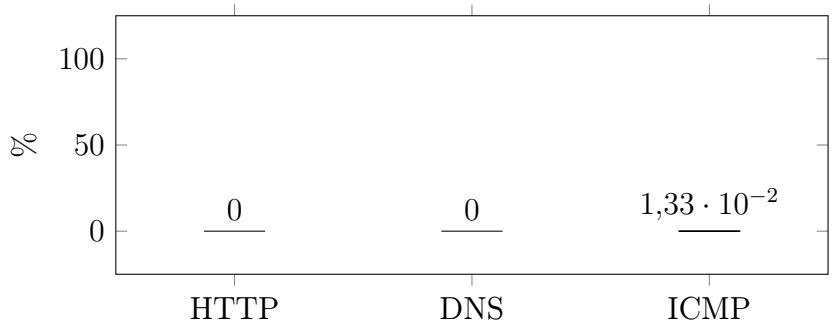
Tím dojte k vytvoření podobné struktury jako u `iptables`:

```
1   # nft -a list table bridge filter
2   table bridge filter {
3           chain input {
4                   type filter hook input priority -200; policy accept;
5           }
6
7           chain forward {
8                   type filter hook forward priority -200; policy accept;
9           }
10
11          chain output {
12                  type filter hook output priority 200; policy accept;
13          }
14  }
```

Nastavení přemostění síťových rozhraní se provede stejně jako v 5.4.3. Tím by bylo prostředí pro nftables připraveno. Ovšem během testování bylo zjištěno, že v jádře 3.16 stabilní verze Debianu nefunguje filtrace pomocí určení cílového portu transportního protokolu. Proto byl z repozitáře „backports“ doinstalován balíček s jádrem verze 4.7, na kterém byl test nftables prováděn.

Testování – bez filtrace

Stejně jako v podsekci 5.4.3 provedeme nejdříve testování bez pravidel. Z grafu 5.7 a tabulky A.7 vidíme, že došlo k zanedbatelnému počtu zahození ICMP zpráv. Ostatní protokoly byly přeneseny 100% úspěšně. Při tomto testu dosáhla datová propustnost průměrné velikosti 617 Mbit/s.



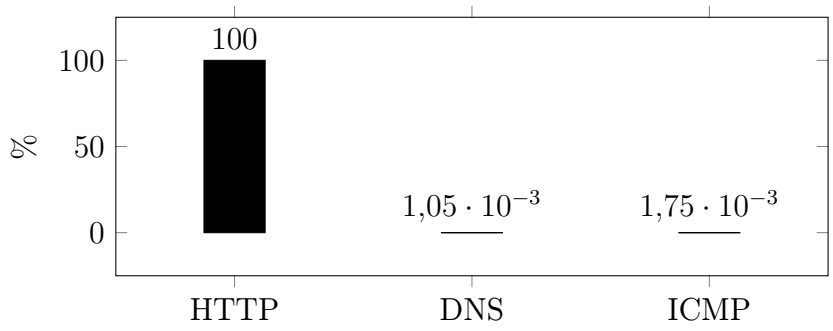
Graf 5.7: Procentuální vyjádření neúspěšných transakcí: nftables – bez filtrace.

Testování – filtrace HTTP

Nyní provedeme filtraci HTTP provozu:

```
1   # nft add rule bridge filter forward ether type ip \
2       tcp dport 80 drop
```

Tento příkaz přidá do tabulky `bridge filter` a řetězce `forward` pravidlo, že pokud bude rámec L2 obsahovat IP provoz (`ether type ip`) a v transportní vrstvě bude přenášen TCP provoz směřující na port 80 (`tcp dport 80`), bude rámec zahozen (`drop`). Graf 5.8 shrnuje očekávaný výsledek (viz také tabulku A.8):



Graf 5.8: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace HTTP.

Testování – filtrace DNS

Podobně jako HTTP vyfiltrujeme i DNS. Pouze určíme, že se jedná o UDP provoz a cílový port je 53:

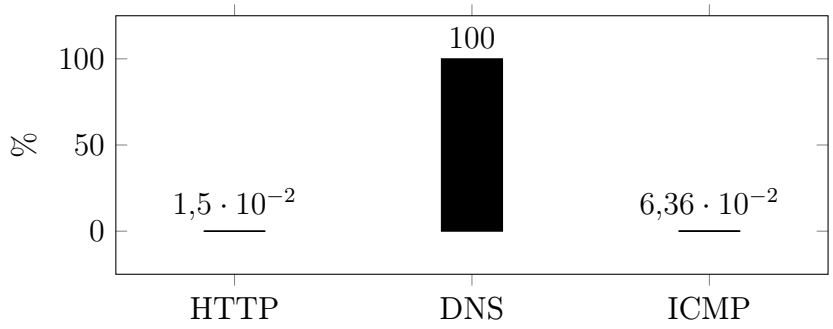
```
1   # nft add rule bridge filter forward ether type ip \
2       udp dport 53 drop
```

Výsledek měření je dle očekávání a je možné ho naleznout v grafu 5.9 a tabulce A.9.

Testování – filtrace ICMP

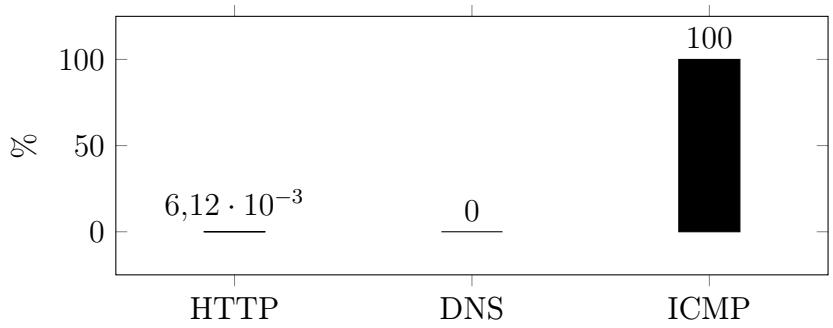
Stejně jako u `iptables` se i zde mírně změní filtrovací příkaz

```
1   # nft add rule bridge filter forward ether type ip \
2       icmp type 8 drop
```



Graf 5.9: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace DNS.

Jak bychom čekali, graf 5.10 dokládá, že i filtrace ICMP byla úspěšná. Z tabulky A.10 lze vyčíst přesné hodnoty.



Graf 5.10: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace ICMP.

Testování – filtrace rozsahu IP adres

Na rozdíl od `iptables`, není potřeba kvůli filtrování rozsahu načítat do nftables žádný modul. Nftables podporují rozsahy adres již v základu:

```
1   # nft add rule bridge filter forward ether type ip \
2           ip saddr {192.168.1.110-192.168.1.200} drop
```

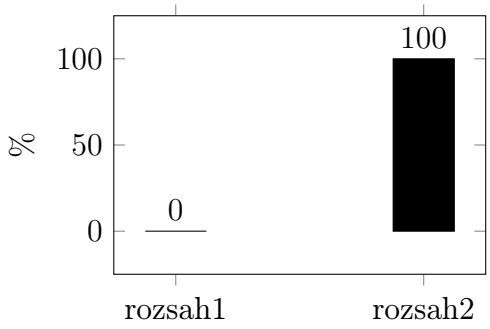
Speciální syntaxí se složenými závorkami definujeme množinu hodnot. Můžeme ji zadat buď jako seznam oddělený čárkami, rozsahem, či kombinací. Po spuštění testu dostaneme výsledek, jenž shrnuje tabulka A.11 a graficky vyjadřuje graf 5.11. Filtrování rozsahu IP adres tedy funguje správně.

5.4.5. PF_RING

Instalace a nastavení

PF_RING již není součástí standardního balíčkovacího systému, a tudíž je nutné stáhnout jeho zdrojové kódy a ručně zkompilovat. Aby mohla být kompilace úspěšná, musíme doinstalovat další požadované balíčky tímto příkazem:

```
1   # apt-get install git build-essential bison \
2       flex linux-headers-$(uname -r) libnuma-dev
```



Graf 5.11: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace rozsahu IP adres.

Nyní můžeme přistoupit ke stažení zdrojových kódů a jejich kompilaci. Stažení provedeme z veřejného repozitáře GitHub:

```
# git clone https://github.com/ntop/PF_RING.git
# cd PF_RING
```

Pro zkompilování zdrojových kódů použijeme příkaz:

```
# make
```

Pokud proběhne kompilace úspěšně, je v adresáři `kernel` vytvořen soubor `pf_ring.ko`, což je jaderný modul, který je pro správnou funkci potřeba načíst:

```
# cd kernel
# insmod pf_ring.ko
# cd ..
```

V tuto chvíli již je možné spouštět PF_RING aplikace. Nás bude nejvíce zajímat program `pfbridge` v adresáři `userland/examples`. Nápověda tohoto prográmku vypadá následovně:

```
# cd userland/examples/
# ./pfbridge -h
pfbridge - Forwards traffic from -a -> -b device using vanilla PF_RING

-h              [Print help]
-v              [Verbose]
-p              [Use pfring_send() instead of bridge]
-a <device>     [First device name]
-b <device>     [Second device name]
-g <core_id>    Bind this app to a core
-w <watermark>  Watermark
```

Jak je z nápovědy patrné, program přeposílá rámce z jednoho rozhraní na druhé. Pokud tedy chceme oboustrannou komunikaci, musíme prográmek pustit dvakrát. To nejjednodušeji provedeme v dalším terminálu. Ačkoliv není v nápovědě zmínka o filtrovací volbě, ze zdrojového kódu této utility lze vyčíst, že rozpoznává ještě argument `-f`, jehož

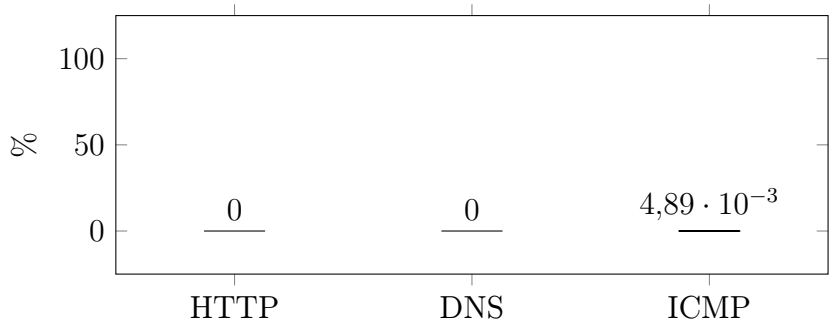
hodnotou může být filtr v BPF syntaxi (např. `tcpdump` používá stejnou syntaxi). Bližší informace o BPF syntaxi lze nalézt v [20]. Při použití volby `-f` je výchozí politikou „zahoď vše“, tzn. že ve filtru musíme explicitně určit to, co má být povoleno. Prostředí je připravené, můžeme přistoupit k samotnému testování.

Testování – bez filtrace

Pro otestování pouhého předávání rámců spustíme ve dvou různých terminálech (ozn. `term1` a `term2`) tyto příkazy:

```
1   (term1)# ./pfbridge -a eth2 -b eth3
2   (term2)# ./pfbridge -a eth3 -b eth2
```

Až na pár nepodstatných zahození je výsledek měření v tabulce A.12 očekávaný (viz graf 5.12). Průměrná rychlost přenosu byla necelých 617 Mbit/s.



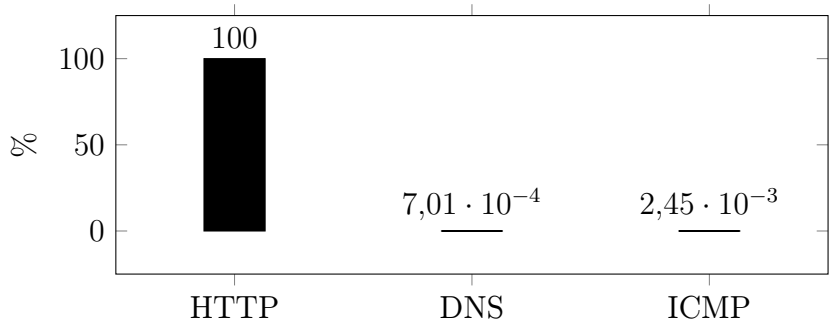
Graf 5.12: Procentuální vyjádření neúspěšných transakcí: PF_RING – bez filtrace.

Testování – filtrace HTTP

Pro filtraci HTTP již musíme použít parametr `-f`:

```
1   (term1)# ./pfbridge -a eth2 -b eth3 -f "not tcp dst port 80"
2   (term2)# ./pfbridge -a eth3 -b eth2 -f "not tcp dst port 80"
```

Notací `"not tcp dst port 80"` říkáme, že má být povoleno vše, co není TCP provoz směřující na port 80. Výsledek v tabulce A.13 je opět spíše očekávaný. Grafické znázornění tabulky ukazuje graf 5.13.



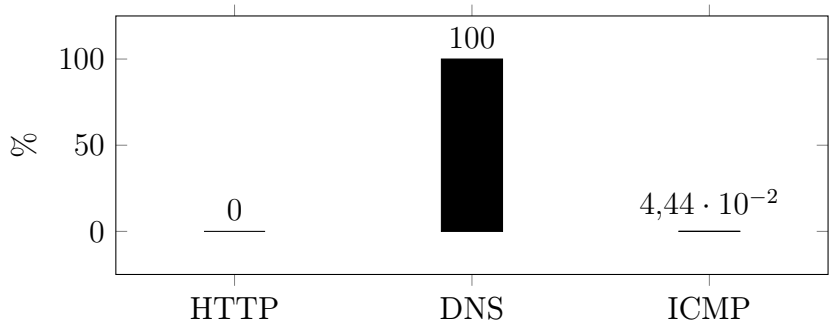
Graf 5.13: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace HTTP.

Testování – filtrace DNS

Podobně jako pro HTTP vytvoříme filtr DNS provozu:

```
1   (term1)# ./pfbridge -a eth2 -b eth3 -f "not udp dst port 53"
2   (term2)# ./pfbridge -a eth3 -b eth2 -f "not udp dst port 53"
```

Výsledek shrnuje tabulka A.14 a graf 5.14.



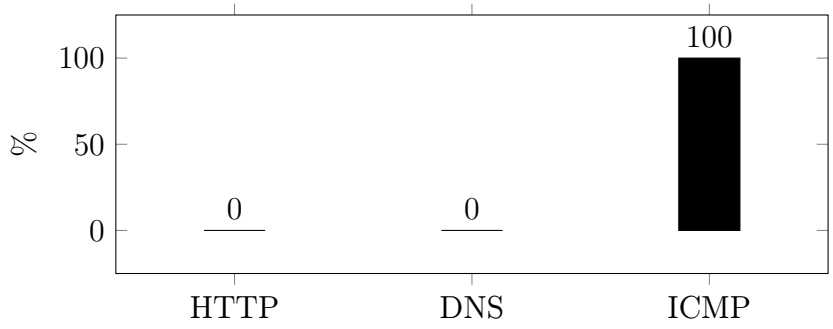
Graf 5.14: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace DNS.

Testování – filtrace ICMP

Jelikož ICMP není transportním protokolem, je i syntaxe filtru mírně odlišná a přizpůsobená požadavkům ICMP:

```
1   (term1)# ./pfbridge -a eth2 -b eth3 -f "not icmp[icmptype]=icmp-echo"
2   (term2)# ./pfbridge -a eth3 -b eth2 -f "not icmp[icmptype]=icmp-echo"
```

Hranatými závorkami ve filtru říkáme, že nás z ICMP hlavičky zajímá pole `icmptype`, a to se musí rovnat konstantě `icmp-echo` (lze použít i dekadickou hodnotu 8). Výsledek tohoto měření je vyjádřen grafem 5.15 a samozřejmě tabulkou A.15.



Graf 5.15: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace ICMP.

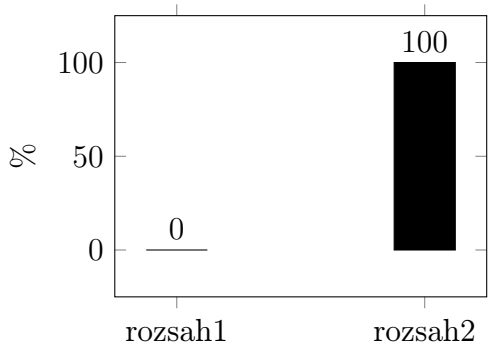
Testování – filtrace rozsahu IP adres

Zadávání rozsahu IP adres, který neodpovídá podsíti, je v BPF syntaxi problém. Proto musel být příkaz složen z několika podmínek:

```
1   (term1)# ./pfbridge -a eth2 -b eth3 -f "not src host 192.168.1.110 \
2                                       and not src host 192.168.1.111 \
```

```
3                                        and not src net 192.168.1.112/28 \
4                                        and not src net 192.168.1.128/26 \
5                                        and not src net 192.168.1.192/29 \
6                                        and not src host 192.168.1.200"
7   (term2)# ./pfbridge -a eth3 -b eth2 -f "not src host 192.168.1.110 \
8                                        and not src host 192.168.1.111 \
9                                        and not src net 192.168.1.112/28 \
10                                       and not src net 192.168.1.128/26 \
11                                       and not src net 192.168.1.192/29 \
12                                       and not src host 192.168.1.200"
```

Celý rozsah2 = `192.168.1.110-192.168.1.200` byl rozdělen na několik samostatných adres a několik podsítí. Samostatné výrazy pak byly spojeny logickou konjunkcí. Ačkoliv není filtr zadán přímočaře, výsledek zobrazený v grafu 5.16 a tabulce A.16 odpovídá požadovanému.



Graf 5.16: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace rozsahu IP adres.

5.4.6. netmap

Instalace a nastavení

Stejně jako PF_RING není ani netmap součástí balíčkovacího systému Debianu a je nutné ho zkompilovat. Požadavky na nainstalované balíky a hlavičkové soubory jsou podobné jako u PF_RINGu (viz podsekce 5.4.5). Pokud jsou tedy nainstalované, můžeme přistoupit ke stažení zdrojových kódů a kompilaci. Zdrojové kódy stáhneme opět z GitHubu:

```
1   # git clone https://github.com/luigirizzo/netmap.git
2   # cd netmap/LINUX
```

V adresáři `LINUX` jsou všechny potřebné kódy pro platformu Linux. Kompilace se provede standardním způsobem:

```
1   # ./configure
2   # make
```

Po kompilaci vznikne v adresáři soubor `netmap.ko`, což je zásuvný modul do jádra, který se stará o komunikaci mezi ovladačem síťové karty a netmap aplikacemi. Načteme jej do jádra:

```
1   # insmod netmap.ko
2   # cd ../../
```

Nyní je potřeba stáhnout a zkompilovat samotný firewall `netmap-ipfw`. Stáhneme jej z GitHubu do stejného adresáře, ve kterém se nachází adresář s netmapem:

```
1   # git clone https://github.com/luigirizzo/netmap-ipfw.git
2   # cd netmap-ipfw
```

Při kompilaci musíme určit cestu k adresáři `sys/` ze zdrojových kódů samotného netmapu:

```
1   # make NETMAP_INC=../netmap/sys
```

Tím dojde k vytvoření binárního souboru `kipfw` a `ipfw/ipfw`. `kipfw` je démon, který se stará o přemostění síťových rozhraní a aplikování nastavených filtrovacích pravidel. `ipfw/ipfw` slouží ke správě pravidel firewallu. Dříve než ale budeme moci firewall spustit, musíme přepnout síťová rozhraní do promiskuitního módu, jinak by komunikace nefungovala:

```
1   # ip link set eth2 promisc on
2   # ip link set eth3 promisc on
```

Následně můžeme spustit firewall démona a předat mu názvy síťových rozhraní, mezi kterými má filtrovat:

```
1   # ./kipfw netmap:eth2 netmap:eth3
```

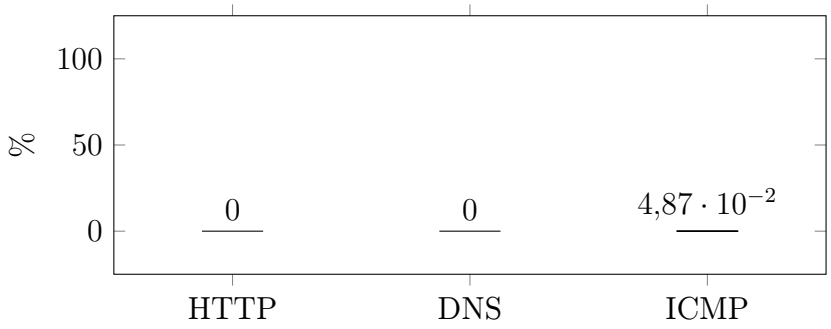
Démona necháme běžet a v jiném terminálu si můžeme prohlédnout aktuální filtrovací tabulku:

```
1   # cd ./ipfw
2   # ./ipfw show
3   65535 0 0 allow ip from any to any
```

To znamená, že poslední pravidlo (s pořadovým číslem 65 535) povoluje vše. Toto je základní instalace a nastavení, aby netmap spolu s `netmap-ipfw` fungoval.

Testování – bez filtrace

Jelikož jsme nechali `kipfw` běžet a výchozí pravidlo povoluje vše, můžeme vyzkoušet propustnost bez filtrace. Až na pár zahození je výsledek zobrazený v grafu 5.17 dle očekávání (viz také tabulku A.17). V tomto případě dosáhla datová propustnost průměrné velikosti 615 Mbit/s.



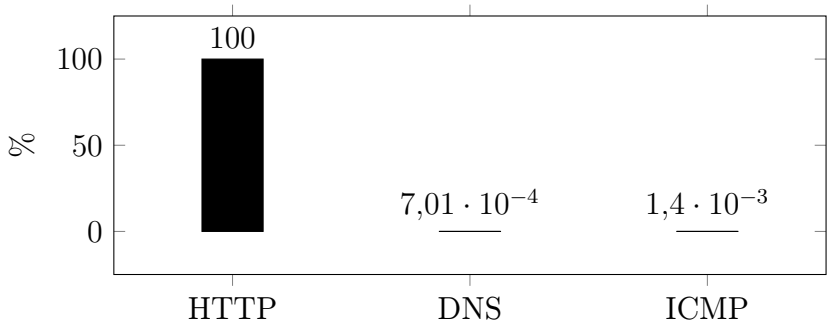
Graf 5.17: Procentuální vyjádření neúspěšných transakcí: netmap – bez filtrace.

Testování – filtrace HTTP

Abychom vyfiltrovali HTTP, musíme přidat pravidlo:

```
1   # ./ipfw add drop tcp from any to any dst-port 80
```

Pravidlo říká, že bude zahozen veškerý TCP provoz z kterékoliv IP adresy (`from any`) mířící na jakoukoliv IP adresu (`to any`) na port 80. Výsledkem měření je graf 5.18 a tabulka A.18.



Graf 5.18: Procentuální vyjádření neúspěšných transakcí: netmap – filtrace HTTP.

Testování – filtrace DNS

Stejně jako u ostatních testovaných nástrojů je syntaxe filtru DNS podobná filtraci HTTP:

```
1   # ./ipfw add drop udp from any to any dst-port 53
```

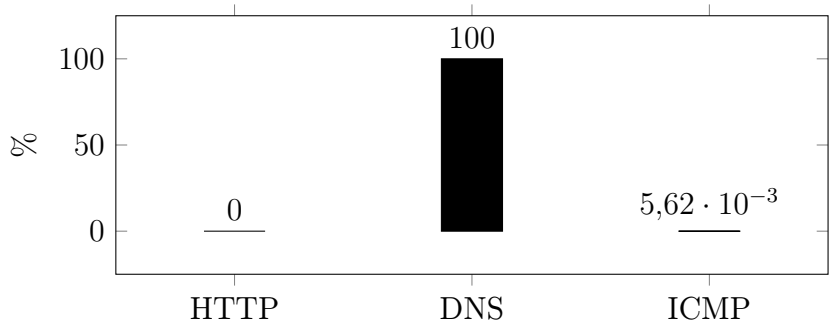
Výsledek měření v tabulce A.19 je znázorněn grafem 5.19.

Testování – filtrace ICMP

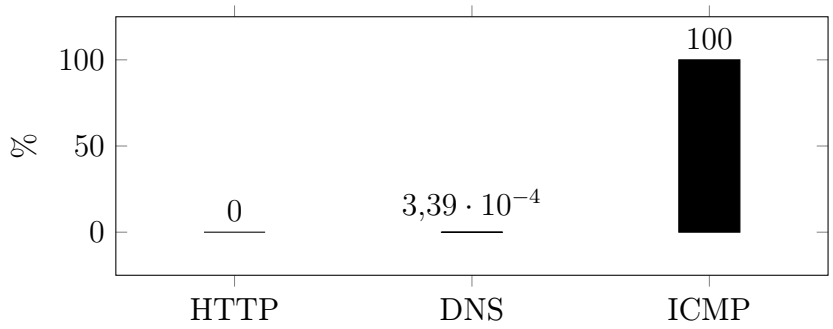
Syntaxe pro filtr ICMP se příliš neliší od filtrace transportních protokolů:

```
1   # ./ipfw add drop icmp from any to any icmptypes 8
```

V tomto případě zadáváme místo cílového portu typ ICMP zprávy. Graf 5.20 dokládá, že i filtrace ICMP byla úspěšná (viz také tabulku A.20).



Graf 5.19: Procentuální vyjádření neúspěšných transakcí: netmap – filtrace DNS.



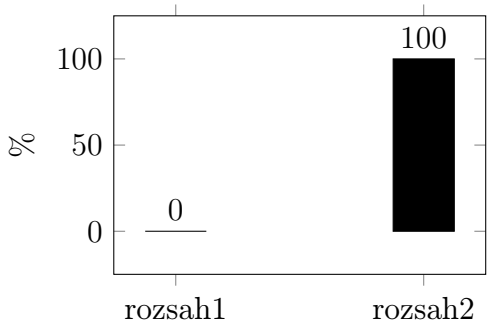
Graf 5.20: Procentuální vyjádření neúspěšných transakcí: netmap – filtrace ICMP.

Testování – filtrace rozsahu IP adres

Ani `netmap-ipfw` neumožňuje zadat přímo rozsah IP adres, ale na rozdíl od PF_RINGu je možné samostatné adresy a podsítě zadat jako seznam oddělený čárkami:

```
1  # ./ipfw add drop ip from 192.168.1.110,192.168.1.111, \
2        192.168.1.112/28,192.168.1.128/26,192.168.1.192/29, \
3        192.168.1.200 to any
```

Opět byl tedy rozsah2 rozdělen na jednotlivé adresy a podsítě a jako seznam zadán do volby `from`. Filtrace IP rozsahu opět fungoval správně (viz tabulka A.21). Grafické znázornění výsledku, které vychází z tabulky A.21, je v grafu 5.21.



Graf 5.21: Procentuální vyjádření neúspěšných transakcí: netmap – filtrace rozsahu IP adres.

5.5. Druhá fáze – testování do rychlosti 10 Gbit/s

Během opakovaného proměřování vybraných nástrojů na rychlosti do 10 Gbit/s byly provedeny naprosto stejné úkony a příkazy, které byly provedeny v první fázi. Proto již dále nebudou tyto činnosti popisovány. Pokud by ovšem nastala nějaká změna, bude v textu samozřejmě popsána.

5.5.1. Síťová topologie

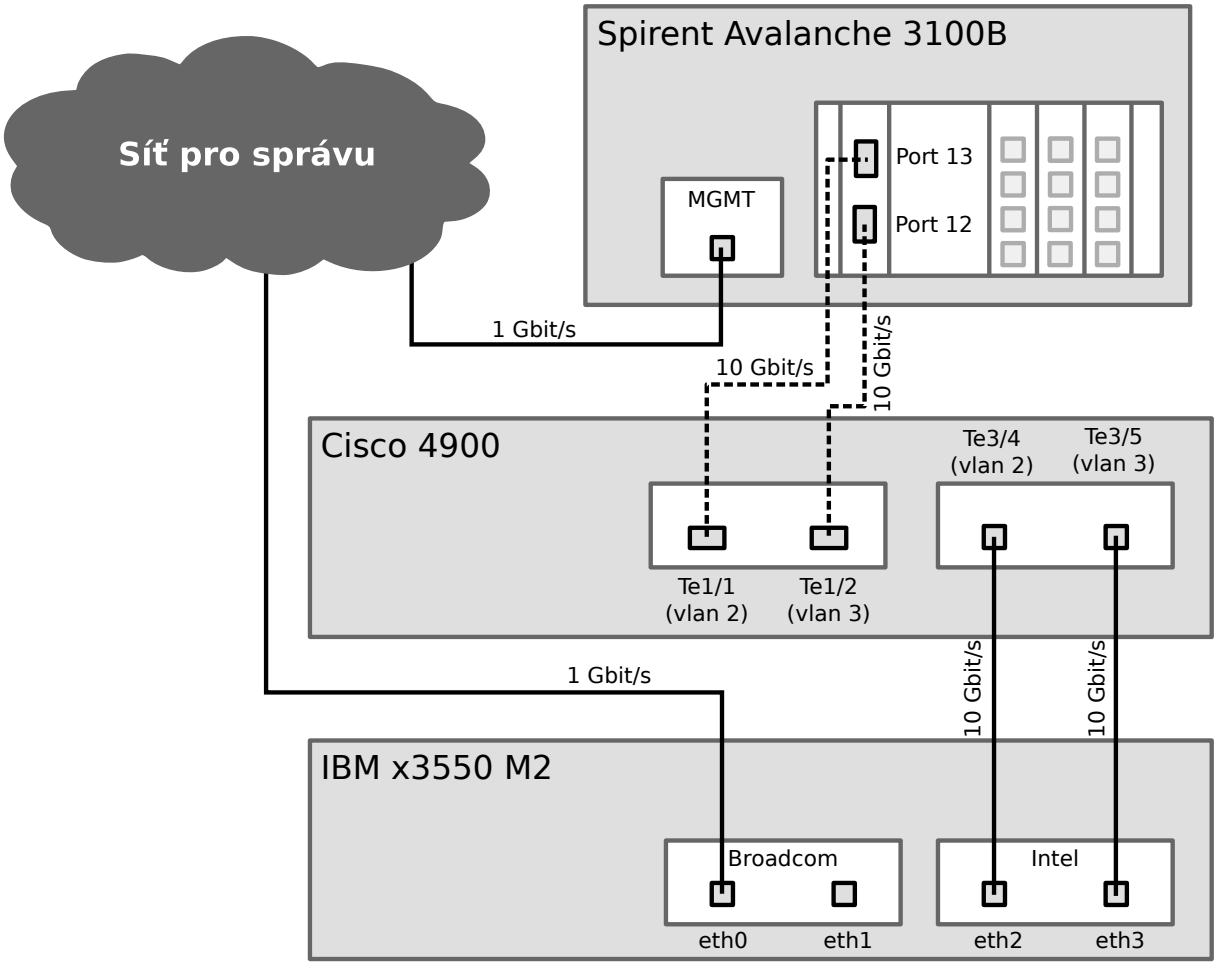
Jelikož má generátor provozu 10Gbit/s rozhraní pouze typu SFP+ s vloženými optickými moduly, bylo nutné vyřešit konverzi signálu z optického na elektrický, protože testovaný server měl k dispozici jen metalická rozhraní. K tomuto účel posloužil přepínač Cisco řady 4900, který nabízí jak SFP+, tak metalické porty pro rychlost 10 Gbit/s. Do něj tedy byly Spirent Avalanche 3100B i testovací server připojeni. Aby bylo zaručeno, že generovaný provoz bude procházet přes testovací server, byly v rámci přepínače vytvořeny dvě virtuální sítě (VLAN), a do každé z nich bylo přidáno jedno rozhraní od obou zařízení. Tím se zaručilo, že provoz se nemohl dostat jinou cestou než přes testovací server. Celou topologii lze zhlédnout na obrázku 5.2. V tomto případě bylo pro generování provozu použito rozhraní 12 a rozhraní 13 pro zpracování provozu.

5.5.2. Nastavení Spirent Avalanche 3100B

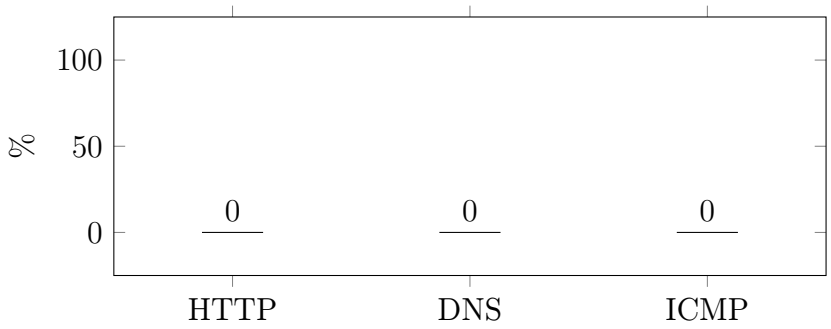
Z pohledu adresování byl Spirent Avalanche 3100B nastaven stejně jako v první fázi testování (viz 5.4.2). Co se ale muselo změnit byla generovaná zátěž, a tedy akce při testu vykonávané. Nastavení testu bylo provedeno tímto způsobem:

- zátěž byla nastavena na 17 000 transakcí za sekundu,
- test probíhal 25 sekund a byl nastaven tak, aby ihned po startu začal generovat nastavenou zátěž,
- akce, které probíhaly během testu, byly následující:
  - 350× `get http://192.168.1.1/index.html` – jedná se o HTTP GET, který vyžaduje stažení 30kB webové stránky nacházející se na daném URL,
  - 15× `ICMP://192.168.1.1 ECHO LENGTH=1472` – odeslání 1472 B dlouhého ICMP Echo požadavku na adresu `192.168.1.1`,
  - 15× `DNS A 192.168.1.1 www.spirent.com` – odeslání DNS dotazu na přeložení domény www.spirent.com na DNS server `192.168.1.1`, jehož odpovědí je jeden A záznam,
- tímto nastavením byl vygenerován provoz o průměrné rychlosti cca 8,4 Gbit/s (referenční měření, viz dále).

Stejně jako v první fázi bylo před samotným testováním provedeno referenční měření. To probíhalo při přímém propojení portu 12 a 13 na zařízení Spirent Avalanche 3100B pomocí optického propojovacího kabelu. Naměřené hodnoty jsou shrnuty v tabulce A.22 a jejich grafické znázornění zobrazuje graf 5.22.



Obrázek 5.2: Tímto způsobem byla zařízení síťově propojena ve druhé fázi. Čárkovaná čára znamená optické propojení, plná čára značí metalické propojení.



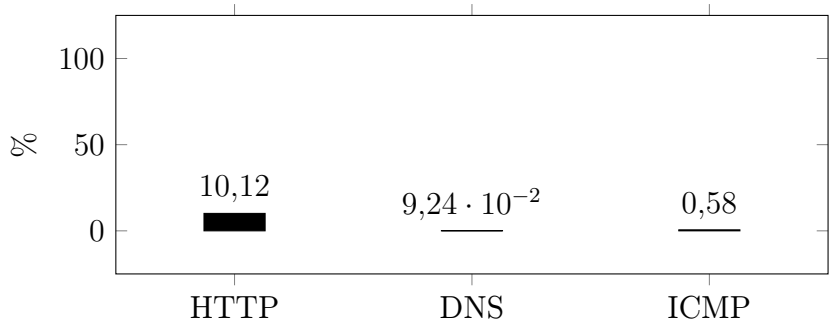
Graf 5.22: Procentuální vyjádření neúspěšných transakcí při referenčním měření na rychlosti 10 Gbit/s. Očekávaně k žádnému neúspěchu nedošlo.

5.5.3. netfilter + iptables

Testování – bez filtrace

Nejdříve bylo provedeno měření bez jakéhokoliv zásahu do filtrovací tabulky. Výsledek je zaznamenán v tabulce A.23 a zobrazen grafem 5.23. V tabulce je vidět, že ačkoliv nebyla filtrace povolena, došlo k docela značnému zahazování provozu (∼5,17 %). To

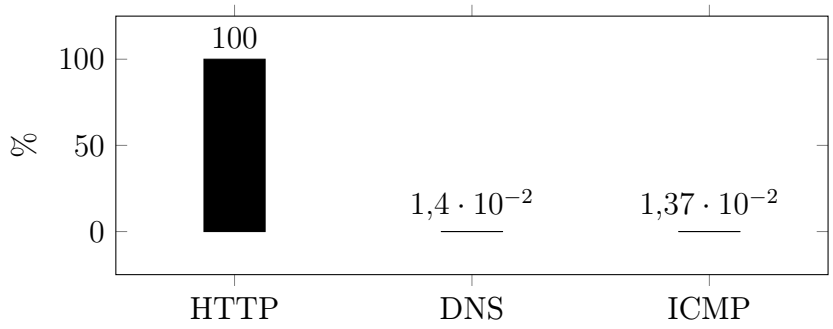
poukazuje na limity přemostění síťových rozhraní pomocí jádra Linux. Při tomto měření bylo dosaženo průměrné datové propustnosti cca 3,49 Gbit/s.



Graf 5.23: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – bez filtrace.

Testování – filtrace HTTP

Výsledek měření při filtraci HTTP zobrazuje graf 5.24 a tabulka A.24.



Graf 5.24: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace HTTP.

Při filtraci HTTP došlo správně k jeho úplnému odstranění z provozu a zároveň došlo k zahození i ostatního provozu. Toto ovlivnění je však velmi malé (setiny procenta). Při tomto, ale i dalších filtracích, nemá cenu uvádět průměrnou rychlost přenosu, jelikož odfiltrováním části provozu musí zákonitě dojít ke snížení datové propustnosti, a tudíž jsou výsledky vůči referenčnímu měření této hodnoty neporovnatelné. Spíše nás zajímá ovlivnění nefiltrovaného provozu.

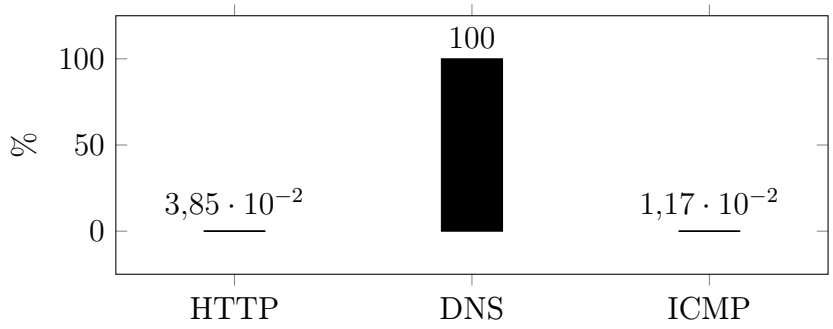
Testování – filtrace DNS

Výsledek tohoto testu vyjadřuje graf 5.25 a tabulka A.25.

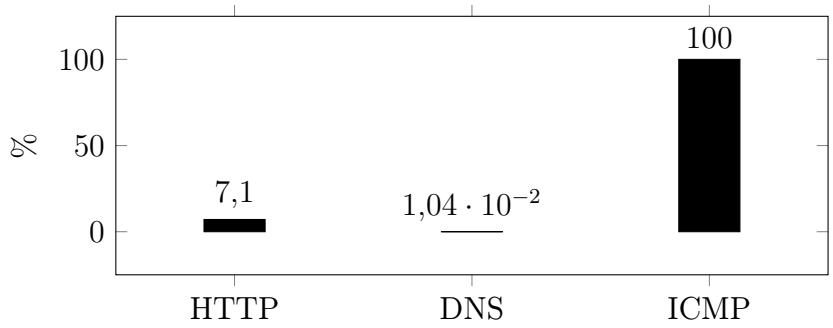
Testování – filtrace ICMP

Tabulka A.26 shrnuje výsledek tohoto měření. Z uvedené tabulky vychází i graf 5.26.

Na tomto měření je vidět, že kromě odstranění ICMP provozu došlo i k nezanedbatelnému zahození HTTP provozu (cca 7,1 %).



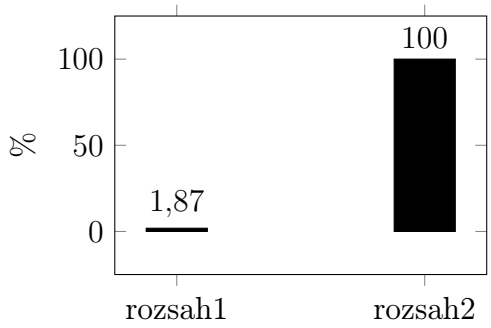
Graf 5.25: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace DNS.



Graf 5.26: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace ICMP.

Testování – filtrace rozsahu IP adres

Výsledek tohoto měření je vyjádřen tabulkou A.27 a grafem 5.27.



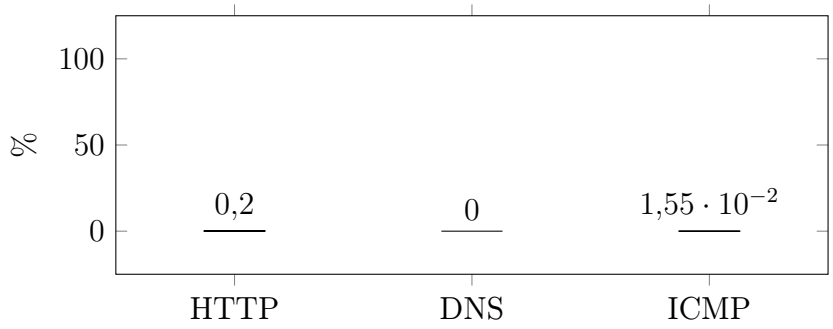
Graf 5.27: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace rozsahu IP adres.

I při tomto měření došlo k celkem velkému ovlivnění HTTP provozu z rozsahu adres, který měl projít beze ztráty.

5.5.4. nftables

Testování – bez filtrace

Opět bylo provedeno nejdříve měření bez jediného filtrovacího pravidla. Výsledek je dán tabulkou A.28 a z ní vycházejícím grafem 5.28.

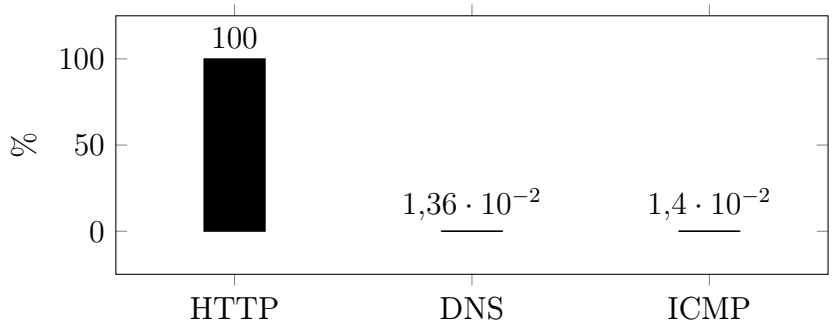


Graf 5.28: Procentuální vyjádření neúspěšných transakcí: nftables – bez filtrace.

Porovnáme-li výsledky s tabulkou A.23 (netfilter + `iptables` – bez filtrace), zjistíme, že došlo k zásadnímu vylepšení výkonu – provedlo se více transakcí, a zároveň došlo ke snížení ztrátovosti. Datová propustnost byla průměrně 6,6 Gbit/s. Toto zlepšení může být způsobeno jak vylepšeními nftables oproti netfilteru + `iptables`, tak novějším jádrem Linux (zdůvodnění použití novějšího jádra pro testování nftables lze nalézt v podsekci 5.4.4).

Testování – filtrace HTTP

Výsledek filtrace HTTP pomocí nftables ukazuje graf 5.29 a tabulka A.29.



Graf 5.29: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace HTTP.

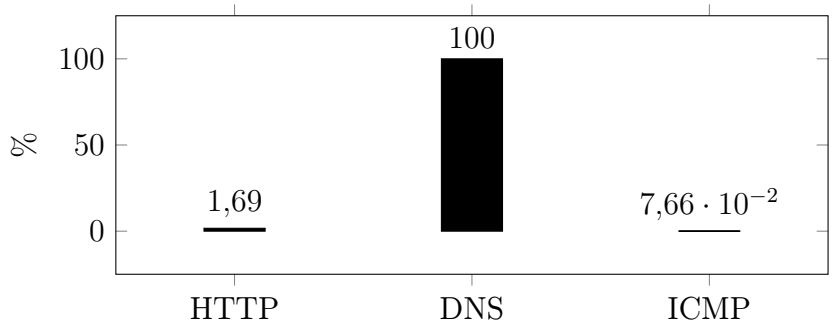
Provedeme-li porovnání s tabulkou A.24, vidíme, že dosažené výsledky jsou v tomto případě srovnatelné. Je to nejspíš z toho důvodu, že odstraněním datově náročného provozu není zátěž tak vysoká, aby si s ní nftables ani netfilter + `iptables` neporadily.

Testování – filtrace DNS

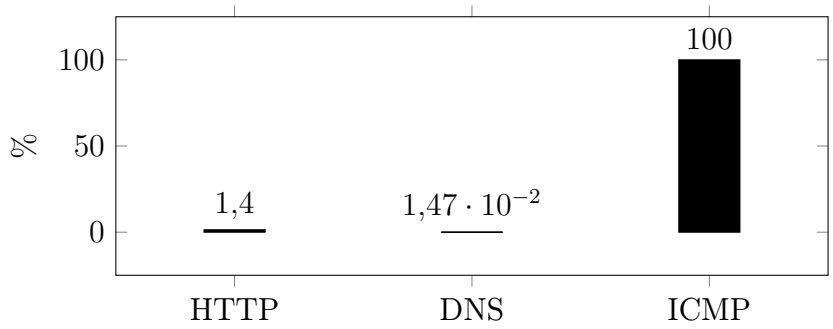
Graf 5.30 shrnuje výsledek filtrace DNS, který je vyjádřen tabulkou A.30. Výsledek je ve srovnání s netfilterem a `iptables` o poznání horší.

Testování – filtrace ICMP

V tomto případě (viz tabulka A.31 a graf 5.31) došlo opět k mírnému zlepšení oproti netfilteru a `iptables` – provedlo se více transakcí a bylo méně neúspěšných transakcí.



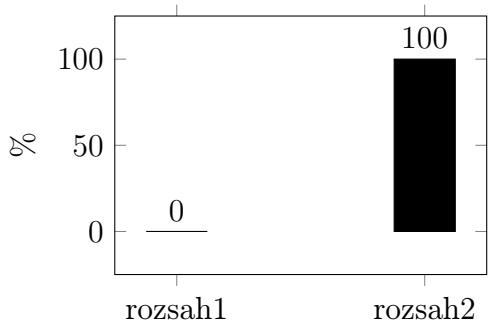
Graf 5.30: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace DNS.



Graf 5.31: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace ICMP.

Testování – filtrace rozsahu IP adres

Výsledek v tabulce A.32 a grafu 5.32 dokazuje, že filtrování rozsahu IP adres v nftables je výkonnější než v případě netfilteru + `iptables`. HTTP a DNS provoz z povoleného rozsahu dokonce nebyl ovlivněn vůbec.



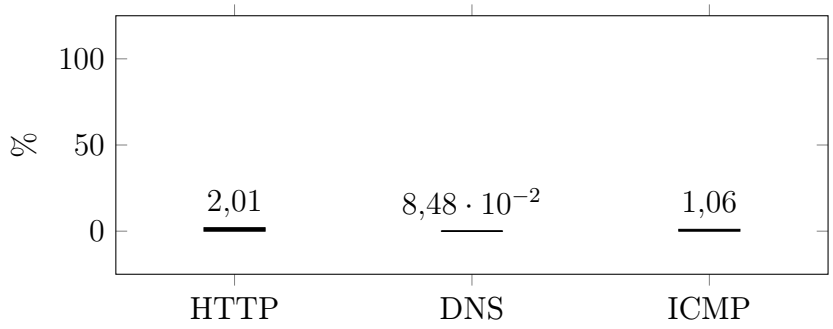
Graf 5.32: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace rozsahu IP adres.

5.5.5. PF_RING

Testování – bez filtrace

Nejdříve bylo provedeno měření (viz tabulka A.33 a graf 5.33) bez použití filtrovacího argumentu.

Srovnáme-li výsledek s předchozími nástroji, zjistíme, že PF_RING je na tom lépe než netfilter + `iptables`. Ačkoliv se provedlo o něco méně transakcí, počet neúspěšných

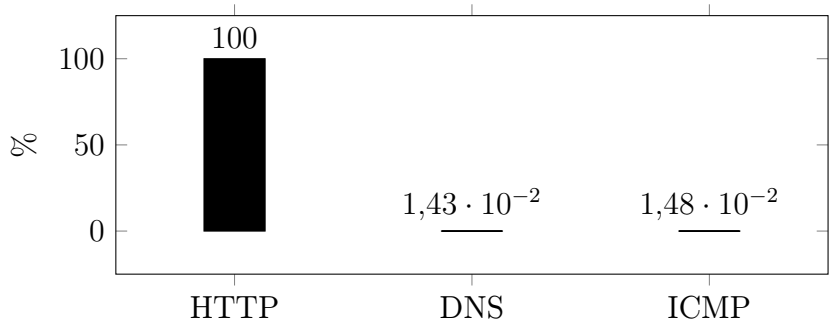


Graf 5.33: Procentuální vyjádření neúspěšných transakcí: PF_RING – bez filtrace.

transakcí je zhruba 4krát menší. Ve srovnání s nftables je však PF_RING daleko horší. Datová propustnost při měření se průměrně pohybovala okolo 4 Gbit/s.

Testování – filtrace HTTP

Výsledek filtrace HTTP shrnuje tabulka A.34 a z ní vycházející graf 5.34.

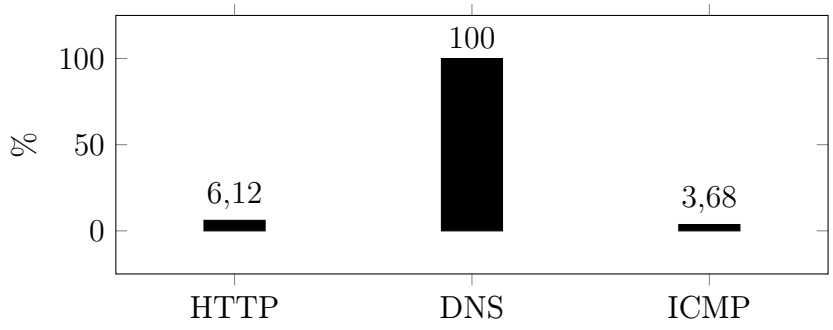


Graf 5.34: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace HTTP.

Filtrace HTTP proběhla úspěšně a výsledky všech třech zatím vyzkoušených nástrojů jsou v zásadě srovnatelné.

Testování – filtrace DNS

Filtrace DNS v podání PF_RINGu nedopadla příliš dobře (viz graf 5.35 a tabulku A.35).

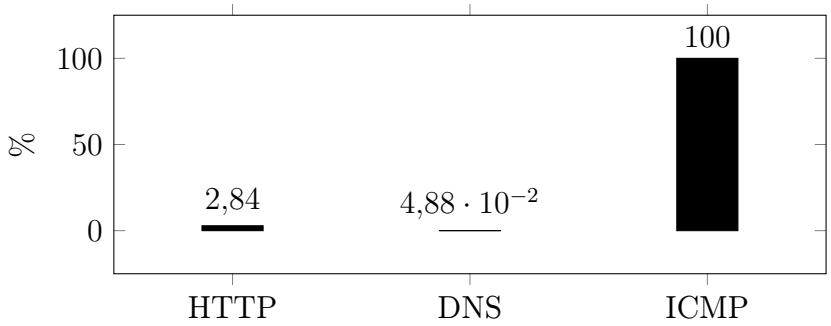


Graf 5.35: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace DNS.

Zcela jistě nejhorší výsledek filtrace DNS z doposud vyzkoušených filtračních nástrojů. Šestiprocentní ztráta HTTP provozu, kdežto ostatní nástroje dosáhly maximálně 1,7% ztráty.

Testování – filtrace ICMP

Výsledek měření filtrace ICMP ukazuje tabulka A.36, a tu shrnuje graf 5.36.

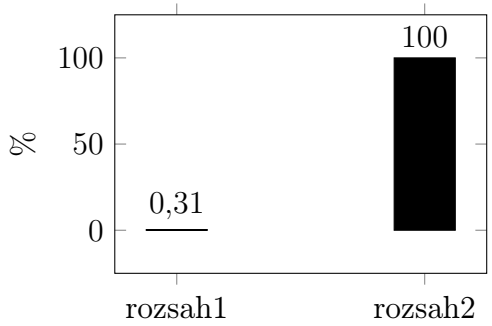


Graf 5.36: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace ICMP.

Porovnáme-li opět výsledky předchozích nástrojů, zjistíme, že si PF_RING stojí lépe než netfilter + `iptables` (asi 3% ztrátovost HTTP provozu oproti 7 %). Ovšem nftables je v tomto případě efektivnější (jen asi 1,4% ztrátovost).

Testování – filtrace rozsahu IP adres

Výsledek posledního testu pro PF_RING je v tabulce A.37 a shrnuje jej graf 5.37.



Graf 5.37: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace rozsahu IP adres.

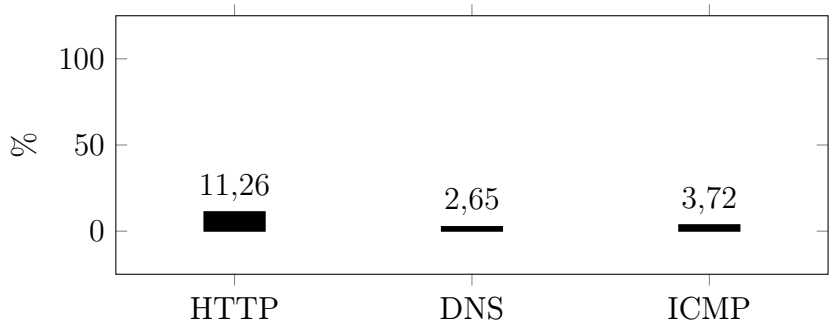
Pouhým srovnáním čísel zjistíme, že při testování netfilteru + `iptables` bylo vyzkoušeno zhruba stejně HTTP transakcí z povoleného rozsahu, avšak cca 9 400 zahození oproti cca 1 400 zahození PF_RINGu je už značný rozdíl. Ovšem nftables byly při tomto testu nejlepší.

5.5.6. netmap

Testování – bez filtrace

I v tomto případě provedeme nejdříve test bez filtrace (viz tabulku A.38).

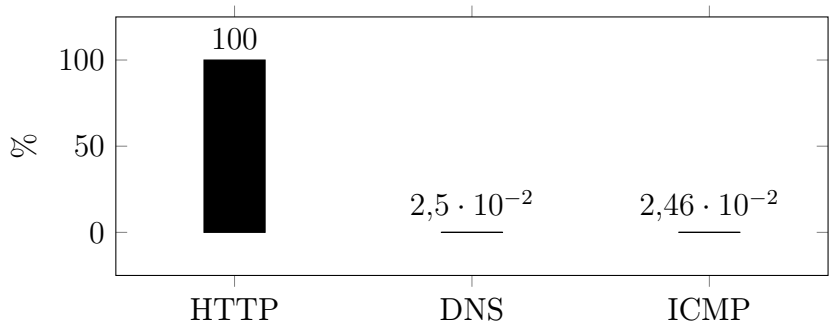
Je zvláštní, že nástroj přímo určený k vysokorychlostní filtraci, dopadne nejhůře z doposud testovaných nástrojů (viz graf 5.38). Celkové zahození 8,8 % transakcí je zcela největší číslo z provedených testů v této fázi testování (samozřejmě bereme v potaz jen stejný test u ostatních nástrojů). Průměrná rychlost přenosu dat se pohybovala okolo 3,6 Gbit/s.



Graf 5.38: Procentuální vyjádření neúspěšných transakcí: netmap – bez filtrace.

Testování – filtrace HTTP

Výsledek měření při filtraci HTTP zobrazuje tabulka A.39.

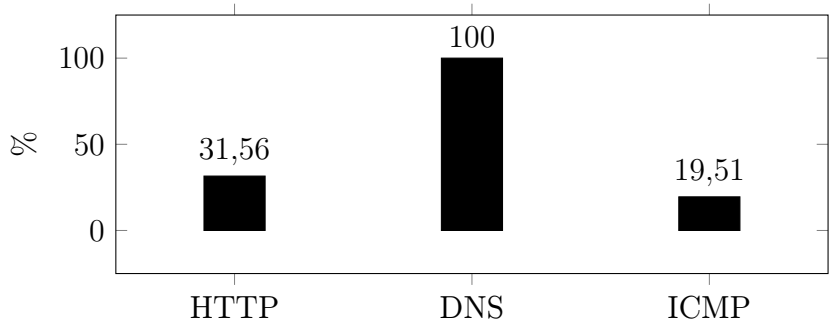


Graf 5.39: Procentuální vyjádření neúspěšných transakcí: netmap – filtrace HTTP.

V porovnání s ostatními nástroji, se při tomto testu vyzkoušelo méně transakcí, ačkoliv počty zahození povolených protokolů jsou stejné. V relativních jednotkách se ale jedná o setiny procent a méně (viz graf 5.39).

Testování – filtrace DNS

Tabulka A.40 shrnuje výsledek filtrace DNS pomocí netmapu. Grafické znázornění tabulky je v grafu 5.40.

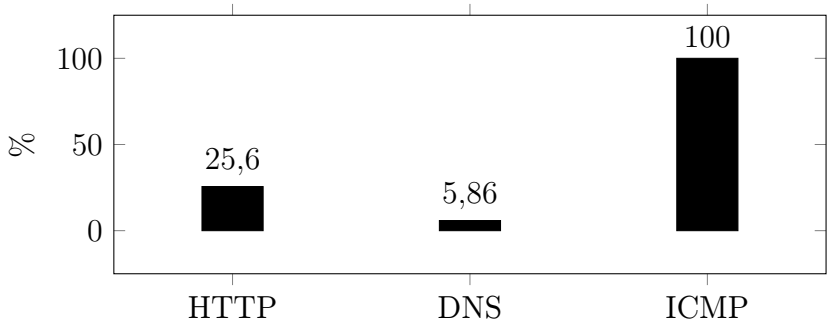


Graf 5.40: Procentuální vyjádření neúspěšných transakcí: netmap – filtrace DNS.

Filtrace DNS je opět nejhorší výsledek z výběru firewallů – ztrátovost HTTP dosáhla k necelým 32 %!

Testování – filtrace ICMP

Tabulka A.41 shrnuje výsledek tohoto měření.

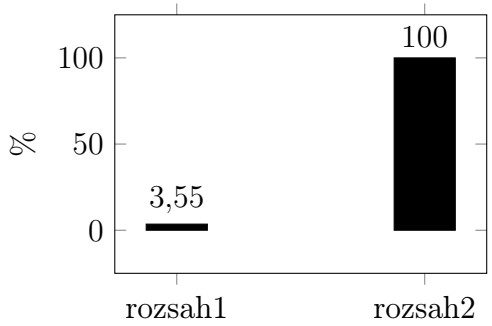


Graf 5.41: Procentuální vyjádření neúspěšných transakcí: netmap – filtrace ICMP.

Opět ne moc dobrý výsledek: $\frac{1}{4}$ HTTP provozu byla zahozena (viz graf 5.41). U ostatních nástrojů je maximum 7 % zahozeného HTTP provozu.

Testování – filtrace rozsahu IP adres

Výsledek posledního testu v tabulce A.42 a grafu 5.42 opět podtrhuje velice slabý výkon netmapu.



Graf 5.42: Procentuální vyjádření neúspěšných transakcí: netmap – filtrace rozsahu IP adres.

Dosažená ztrátovost provozu z povoleného rozsahu dosahuje cca 3,55 procent. I u netfilteru s `iptables` to bylo jen asi 1,87 %.

Ačkoliv výsledky netmapu nejsou příliš dobré, je zde možnost je vylepšit použitím upravených ovladačů síťových karet speciálně pro spolupráci s netmapem. Tyto ovladače jsou distribuovány v rámci zdrojových kódů netmapu a jsou dostupné především pro síťové karty Intel. V první fázi testování nebylo nutné použít upravené ovladače, jelikož výkon standardních ovladačů byl dostatečný pro dané použití. Nyní se však tato možnost nabízí.

5.5.7. netmap s vlastními ovladači NIC

Instalace a nastavení

Upravené ovladače by se měly zkompilovat zároveň s netmapem, tudíž by již měly být připraveny z první fáze testování. Nalézt je můžeme v adresáři `netmap/LINUX/[název`

`ovladače]`. V našem případě je to adresář `netmap/LINUX/ixgbe`, jelikož `ixgbe` je název ovladače. Ovladač není nic jiného než jaderný modul, který je potřeba nahrát do jádra. Nejdříve je však nutné vyjmout z jádra modul se standardním jaderným ovladačem:

```
1   # rmmod ixgbe
```

Následně je možné vložit do jádra upravený ovladač. Je třeba upozornit, že je nutné mít již načtený jaderný modul netmapu. Jelikož však byl upravený ovladač novější a s novými funkcemi než standardní ovladač, bylo v tomto případě potřeba nejdříve načíst ještě jeden modul „vxlan“ (pro nás zcela nesouvisející):

```
1   # modprobe vxlan
2   # cd netmap/LINUX/ixgbe
3   # insmod ixgbe.ko
```

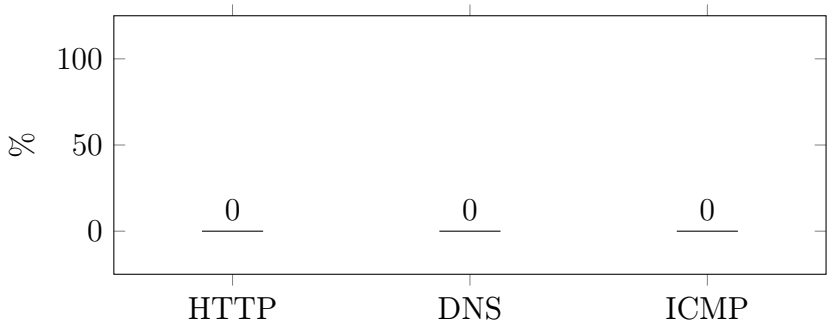
Tím by měla být instalace upravených ovladačů dokončena. Správnost lze zkontrolovat následujícím příkazem, který by měl vrátit podobný výpis:

```
1   # lsmod | grep netmap
2   netmap                126169  1 ixgbe
```

Důležitý je řádek 2, který říká, že modul `netmap` o velikosti 126 169 bajtů v paměti je používaný jedním modulem `ixgbe`.

Testování – bez filtrace

Jako obvykle provedeme nejdříve měření bez filtračních pravidel. Výsledek znázorňuje graf 5.43 a tabulka A.43.

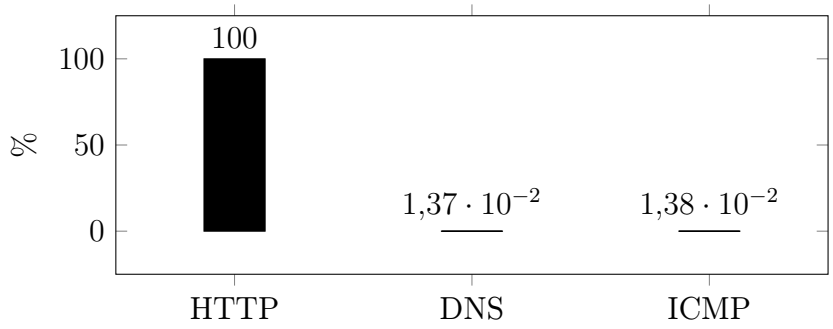


Graf 5.43: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – bez filtrace.

Porovnáme-li hodnoty s netmapem bez speciálních ovladačů, zjistíme, že došlo k velkému zvýšení výkonu. Dokonce nedošlo k jediné neúspěšné transakci! Celkový počet transakcí je jen o několik málo tisíc menší než referenční měření. I průměrná datová propustnost je srovnatelná: ∼8,3 Gbit/s!

Testování – filtrace HTTP

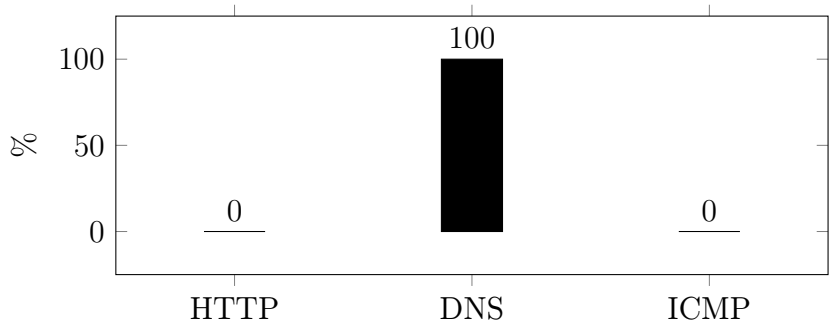
Výsledek tohoto měření zobrazuje graf 5.44 zároveň s tabulkou A.44. Hodnoty tohoto testu jsou srovnatelné u všech testovaných nástrojů.



Graf 5.44: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace HTTP.

Testování – filtrace DNS

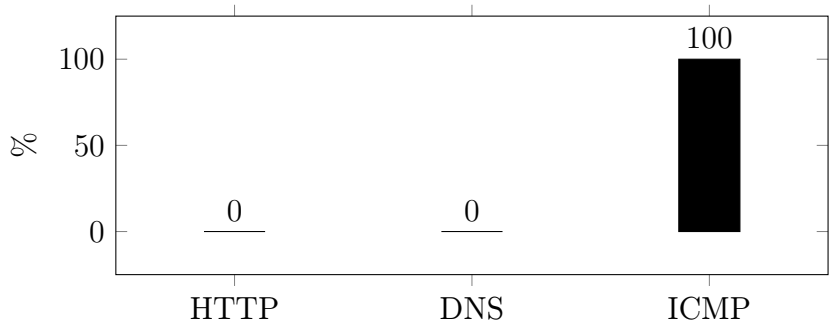
Tabulka A.45 shrnuje výsledek filtrace DNS pomocí netmapu s upraveným ovladačem a graf 5.45 dokazuje, že filtrace DNS pomocí této kombinace nástrojů je jednoznačně nejvýkonnější, jelikož zde nedošlo k žádnému nechtěnému zahození. U všech ostatních nástrojů k nějakému minimálnímu zahození došlo.



Graf 5.45: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace DNS.

Testování – filtrace ICMP

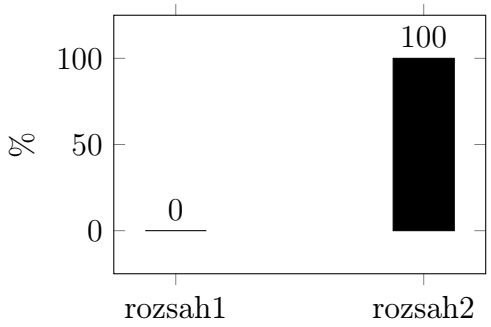
V tomto případě (viz tabulka A.46) byla filtrace opět provedena bez jakéhokoliv nechtěného zahození, jak dokazuje graf 5.46.



Graf 5.46: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace ICMP.

Testování – filtrace rozsahu IP adres

I poslední výsledek měření v tabulce A.47 a grafu 5.47 dokazuje, že netmap (respektive `netmap-ipfw`) ve spojení s upraveným ovladačem síťové karty, by mohl být ideální platformou pro vysokorychlostní filtraci dat na běžně dostupném hardwaru. Opět nedošlo k žádnému (anebo minimálnímu) nechtěnému zahození provozu, který pocházel z povoleného rozsahu (lze důvodně předpokládat, že zahozený ICMP provoz pocházel ze zablokovaného rozsahu – počet neúspěšných ICMP transakcí zhruba odpovídá počtu neúspěšných DNS transakcí).



Graf 5.47: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace rozsahu IP adres.

5.6. Vyhodnocení výsledků 1. a 2. fáze

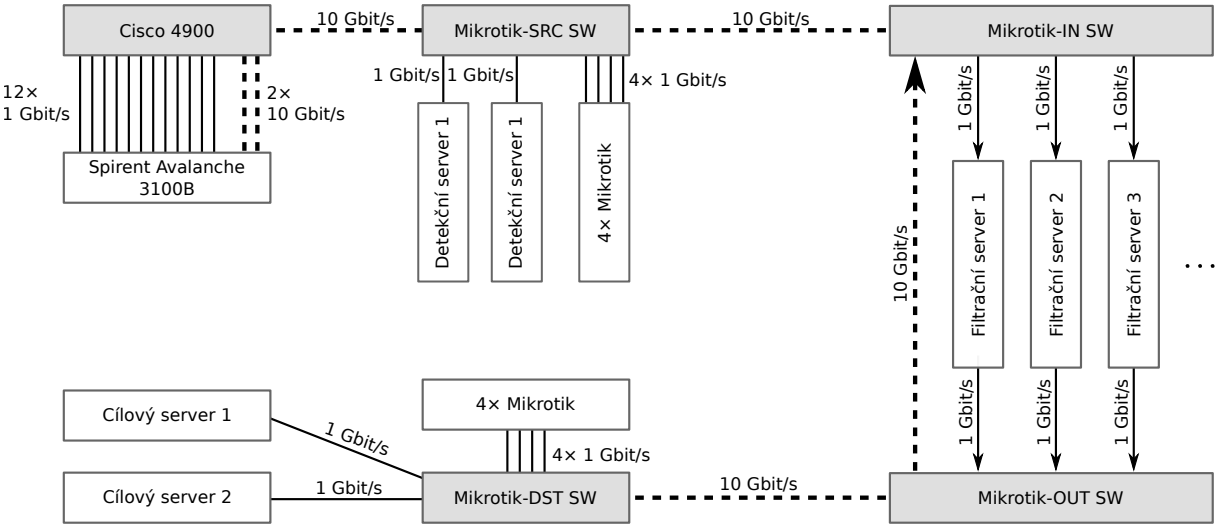
Výsledky měření v první fázi testování byly v drtivé většině případů srovnatelné a dle očekávání. Výjimkou byla pouze filtrace DNS, kdy si netfilter a `iptables` příliš neporadily s ostatním provozem a v docela velké míře zahazovaly HTTP a ICMP transakce. Ve většině případů se také vyskytovala nechtěná zahození, ovšem ve velmi malé míře (setiny procenta). Při pozorování průběhu měření bylo zjištěno, že tato zahození se vyskytovala ihned po spuštění testu. Je tedy možné, že budou tyto ztráty souviset s výchozím režimem, ve kterém se ovladač síťové karty nachází. Jak již bylo popsáno v sekci 3.1, je zpočátku ovladač v režimu, kdy posílá požadavky na přerušení. Až při určitém zatížení přechází do režimu pollingu. A zřejmě právě v době, než ovladač přejde z režimu přerušení do režimu pollingu, může dojít k určitým ztrátám, jelikož byl test nastaven tak, aby generoval danou zátěž ihned po startu testu.

Druhá fáze měření byla již zajímavější. Z logiky věci bylo jasné, že filtrace téměř 10Gbit/s provozu nebude tak snadná jako filtrace provozu do rychlosti 1 Gbit/s, a s největší pravděpodobností bude docházet k nevynuceným ztrátám. Což se záhy potvrdilo. Z naměřených výsledků zcela nevyplývá, že by na tom jádro Linux společně s netfilterem a `iptables` byly výkonnostně příliš špatně. V některých testech dopadlo spojení netfilter + `iptables` lépe než PF_RING a prakticky ve všech testech bylo lepší než netmap s jaderným ovladačem. Ten v této konfiguraci propadl ve všech testech. Naproti tomu velice dobře dopadly nftables. Až na filtraci DNS byly výsledky ze všech testovaných nástrojů se standardními linuxovými ovladači nejlepší. I tak byla propustnost oproti referenčnímu měření o téměř 2 Gbit/s nižší. Jako nejlepší se při testech ukázal netmap ve spojení se speciálně upravenými ovladači. Drtivá většina testů byla úspěšná, tedy testy proběhly

bez jediného nevynuceného zahození. Také bylo dosaženo datové propustnosti srovnatelné s referenčním měřením.

5.7. Testování v experimentální síti

V rámci výzkumu a vývoje detekčních a filtračních schopností systémů pro ochranu před DDoS útoky, vzniklo na půdě Fakulty elektrotechniky a komunikačních technologií VUT experimentální pracoviště, skládající se z uzavřené sítě, v níž je možné testovat a simulovat různé druhy síťového provozu a kybernetických útoků. Jako generátor provozu slouží Spirent Avalanche 3100B, který byl již popsán v sekci 5.2. Dále jsou součástí sítě dva servery, které fungují jako cíle generovaného provozu. Hlavními prvky experimentální sítě jsou pak dva servery pro detekci útoků a osm filtračních serverů. Pro síťovou infrastrukturu jsou využity hlavně přepínače firmy Mikrotik. Páteř sítě je propojena optickými propoji a funguje na rychlosti 10 Gbit/s, ovšem jednotlivé servery jsou připojeny k 1Gbit/s rozhraním. Součástí sítě jsou i další zařízení Mikrotik, které slouží pouze ke generování šumového provozu. Všechny důležité prvky sítě jsou pak připojeny do zvláštního síťového segmentu, který slouží ke správě zařízení a je možné skrz něj přistupovat do Internetu. Topologie celého experimentálního pracoviště je zobrazena na obrázku 5.3.



Obrázek 5.3: Topologie sítě experimentálního pracoviště. Plnou čarou jsou metalické propoje, čárkovanou čárou jsou optické propoje.

Aby bylo zaručeno, že určitý provoz bude procházet přes správný filtrovací server, je na přepínači, ke kterému jsou filtrační servery připojeny, nastaveno, aby na základě určité cílové IP adresy posílal provoz určitým síťovým rozhraním. Toto nastavení je provedeno jen pro provoz ve směru Spirent Avalanche 3100B → cílové servery. V opačném směru jsou rámce předávány přímo mezi přepínači bez účasti filtračních serverů. Díky tomuto nastavení může v síti pracovat paralelně více studentů bez toho, aby se výrazným způsobem vzájemně ovlivňovali.

5.7.1. Testovací server – HP ProLiant DL380

Původní testovací server IBM System x3550 M2 z první a druhé fáze nebyl do experimentální sítě zapojen, ale místo toho byl využit server HP ProLiant DL380 (všechny filtrovací servery byly tohoto typu). Server obsahuje dva jednojádrové 64bitové procesory Intel Xeon pracující na frekvenci 3,40 GHz. K dispozici má 8 GB operační paměti a pevné disky mají dohromady kapacitu 218 GB. Z pohledu síťových možností připojení se na základní desce serveru nacházejí dvě síťová rozhraní Broadcom Corporation NetXtreme BCM5704 Gigabit Ethernet. Jedno z těchto rozhraní bylo využito pro připojení k síti pro správu operačního systému, druhé zůstalo nevyužito. Dále byla do počítačového systému vložena rozšiřující karta s dalšími dvěma síťovými rozhraními. Tato rozhraní byla typu Intel Corporation 82546EB, a každé z nich nabízí přenosovou rychlost až 1 Gbit/s. V linuxovém jádře využívá ovladače `e1000`. Obě rozhraní z rozšiřující karty byla využita pro připojení k experimentální síti. Operačním systémem na tomto serveru byl Debian Jessie s jádrem Linux ve verzi 4.9.

Pro vyzkoušení filtrace na agregovaných linkách byla ze serveru původní Intel karta vyjmuta a nahrazena čtyřportovou verzí stejné značky i typu. Agregovaly se tedy dvě a dvě 1Gbit/s linky (viz další text).

5.7.2. Koordinace práce s experimentální sítí

Jak již bylo uvedeno v úvodu této sekce, s experimentální infrastrukturou může paralelně pracovat více uživatelů. Jelikož jsou prostředky sdílené a jedná se o čistě přepínanou síť, bylo každému uživateli přiděleno jedno gigabitové rozhraní na generátoru Spirent Avalanche 3100B, dále dostal přidělený blok šesti zdrojových IP adres, ze kterých Avalanche odesílá provoz, a nakonec dvě cílové IP adresy, které byly nastaveny na cílových serverech (na každém cílovém serveru jedna). Na základě těchto cílových adres byly na přepínači Mikrotik-IN (viz obrázek 5.3) nastavena pravidla, podle kterých přepínal odchozí provoz přes jednotlivé filtrační servery. Prostředky alokované pro vypracování této práce:

- gigabitové rozhraní číslo 4 na Spirent Avalanche 3100B,
- zdrojové IP adresy: `192.168.2.174-192.168.2.179`,
- cílové IP adresy: `192.168.2.143` a `192.168.2.153`.

5.7.3. Organizace a způsob testování

Podobně jako při předchozím testování filtrovacích nástrojů v malé, uzavřené a jednoúčelové síti, bylo i testování v síti experimentálního pracoviště rozděleno do dvou fází. V první fázi probíhalo testování opět na rychlosti přenosu dat do 1 Gbit/s. V druhé fázi byla ve filtračním serveru vyměněna síťová karta za model s více 1Gbit/s síťovými rozhraními a síť upravena tak, aby mohla být odzkoušena agregace síťových rozhraní. V tomto případě byl tedy server testován na rychlosti do 2 Gbit/s.

Testovací scénář první fáze byl na základě požadavků zadání práce zvolen takto: každý nástroj byl podroben několika testům a každý test proběhl pro dvě různé přenosové rychlosti. Jelikož prochází přes filtrační server primárně jen provoz z generátoru na cílové servery, musí být přenosová rychlost regulována pomocí požadavků, které generátor odesílá. Nižší testovací rychlost byla z největší části tvořena nejmenšími možnými packety.

Tabulka 5.2 ukazuje procentuální zastoupení délek packetů při nižší přenosové rychlosti. V tomto nastavení bylo bez filtrace dosaženo datové propustnosti průměrně cca 176 Mbit/s a generátor posílal packety průměrnou rychlostí zhruba 335 000 packetů/s.

Akce prováděné generátorem při vytváření síťového provozu nižší přenosové rychlosti byly tyto:

- `get http://192.168.2.143/index2.html`
  `get http://192.168.2.153/index2.html` – provede HTTP GET na oba cílové servery, který vyžaduje stažení prázdné webové stránky nacházející se na daném URL (požadavek i odpověď tedy obsahují jen běžné hlavičky),

- `DNS A 192.168.2.143 localhost`
  `DNS A 192.168.2.153 localhost` – odeslání DNS dotazu na A záznam domény „localhost“ na cílové servery `192.168.2.143` a `192.168.2.153`,

- 351× `ICMP://192.168.2.143 ECHO LENGTH=18`
  351× `ICMP://192.168.2.153 ECHO LENGTH=18` – odeslání ICMP Echo dotazu opět na oba cílové servery (délka 18 bajtů je délka jen datové části, se všemi hlavičkami vychází délka rámce na 64 bajtů).

Velikost packetu (bajty)	**Procentuální zastoupení (%)**
0–64	98,74
65–128	0,63
257–512	0,63

Tabulka 5.2: Rozložení velikostí packetů při nižší přenosové rychlosti (cca 176 Mbit/s).

Vyšší přenosová rychlost, při které byly nástroje testovány, byla naopak tvořena z větší části packety, jejichž velikost přesahovala 1 024 bajtů. Zastoupení jednotlivých velikostí v datovém toku lze najít v tabulce 5.3. Při nastavení této datové propustnosti bylo bez filtrace dosaženo průměrně asi 775 Mbit/s odesílaných dat při průměrných cca 88 000 packetů za sekundu.

Akce, které generátor prováděl, byly následující:

- `post http://192.168.2.143/<POST_FILE="img0" CONTENT_TYPE="image/jpg">`
  `post http://192.168.2.153/<POST_FILE="img0" CONTENT_TYPE="image/jpg">`
  – provede HTTP dotaz metodou POST na oba cílové servery, ve kterém odešle 1,4 kB velký soubor (`img0` je označení daného souboru v softwaru pro ovládání generátoru),

- `DNS A 192.168.2.143 localhost`
  `DNS A 192.168.2.153 localhost` – stejný význam jako u nižší přenosové rychlosti,

- 65× `ICMP://192.168.2.143 ECHO LENGTH=1200`
  65× `ICMP://192.168.2.153 ECHO LENGTH=1200` – odeslání ICMP Echo dotazu (s délkou datové části 1 200 bajtů) na oba cílové servery.

Na obrázku 5.4 je možné zhlédnout společné nastavení zátěže a časového průběhu pro nižší rychlost. Pokud byla testována rychlost vyšší, tak se kromě samotných akcí měnila i zátěž z 1 060 na 580 simulovaných uživatelů za sekundu (SimUsers/second). Jeden test trval celkově 67 sekund, ovšem plná zátěž byla generována jen 30 sekund. Prvních 30

Velikost packetu (bajty)	Procentuální zastoupení (%)
0–64	10,46
65–128	1,38
129–256	1,32
1 025–1 500	85,52
1 500+	1,32

Tabulka 5.3: Rozložení velikostí packetů při vyšší přenosové rychlosti (cca 775 Mbit/s).

sekund nebyl generován žádný provoz, protože bylo zjištěno, že během této půl minuty dochází k velkému počtu zahozených spojení. S největší pravděpodobností je to způsobeno tím, že Spirent Avalanche 3100B zprovozní testovací port jen na dobu testování, a pak jej zase vypne. Přepínač Cisco 4900, ke kterému byl generátor připojen, měl nastaveny porty tak, aby po aktivaci portu necelých 30 sekund vyčkával a přijímal a odesílal jen rámce spanning tree protokolu (protokol pro eliminaci smyček v přepínané síti) a ostatní provoz zahazoval (lze tak usoudit na základě zachycené komunikace). Zbylých 7 sekund je pak tzv. „ramp down“ fáze, která slouží k uvolnění alokovaných prostředků pro spojení.

Nastavení filtračních nástrojů při jednotlivých testech bylo následující:

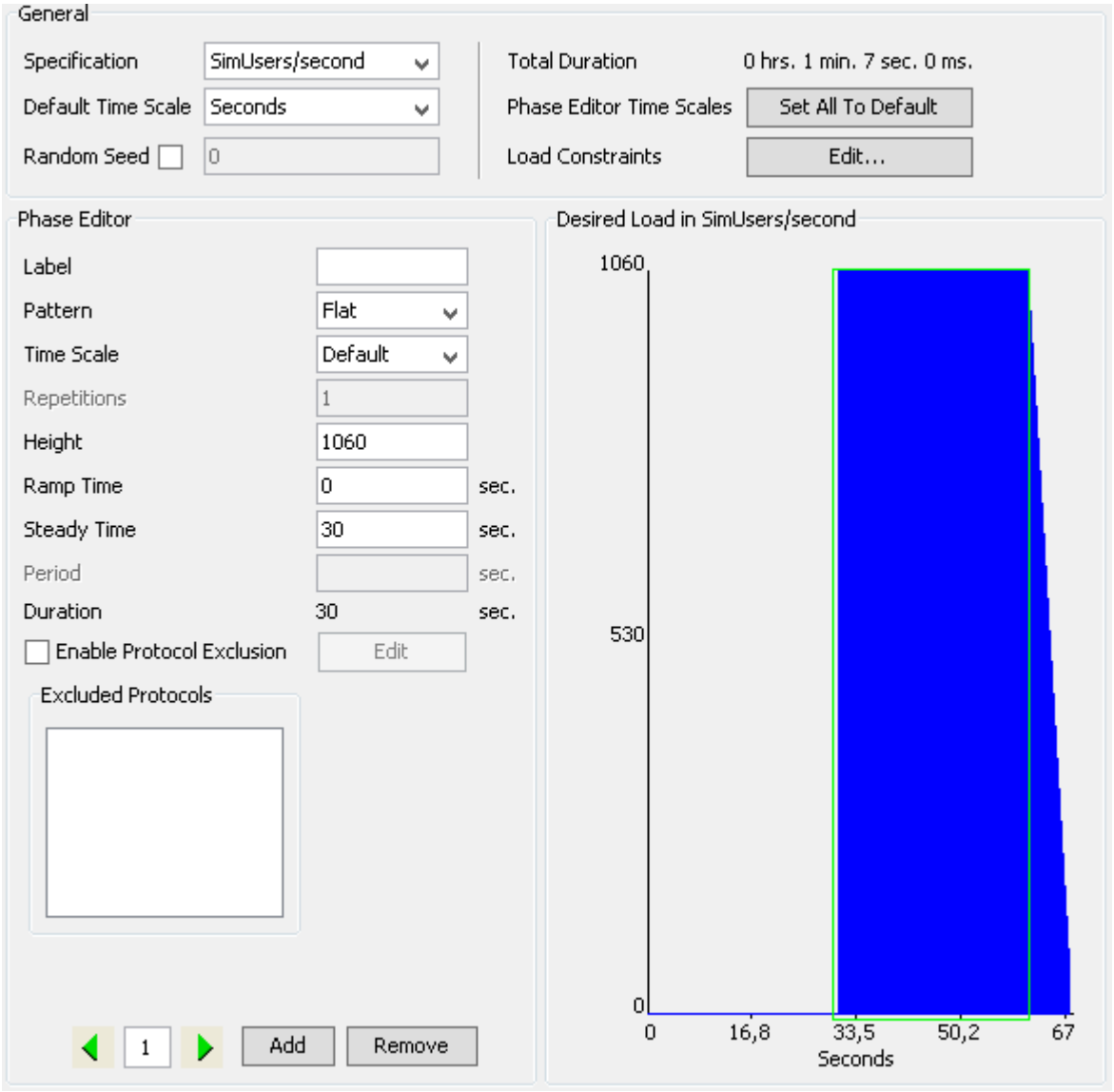
1. bez filtrace,
2. pravidlo pro filtraci na základě zdrojové IP adresy + pravidlo pro filtraci na základě cílové IP adresy,
3. pravidlo pro filtraci na základě zdrojového portu transportní vrstvy + pravidlo pro filtraci na základě cílového portu transportní vrstvy,
4. 100 nevýznamných pravidel pro filtrování zdrojové IP adresy (z jiného IP rozsahu) + pravidlo pro filtraci veškerých TCP komunikací + pravidlo pro filtraci veškerých UDP komunikací.

U těch nástrojů, které to umožňují, by poslední test proběhl i pro nastavení s hlídáním stavu spojení, kdy by první pravidlo povolilo již dříve povolená spojení. Jako transportní protokol pro třetí test byl zvolen UDP, jelikož nepotřebuje sestavit spojení před přenosem dat, což je v této situaci výhodnější. Každé nastavení filtrace pak bylo vyzkoušeno při obou dříve zmíněných rychlostech.

Aby bylo možné sledovat správné nastavení filtrace a případné ovlivňování dalšího provozu, byl rozsah přidělených IP adres v softwaru pro definování testů rozdělen na tři části:

- `192.168.2.174` (dále označen jako rozsah1),
- `192.168.2.175` (dále označen jako rozsah2),
- `192.168.2.176-192.168.2.179` (dále označen jako rozsah3).

První dvě adresy byly použity ve filtračních pravidlech, zbylé čtyři adresy zůstaly povoleny a sloužily pro generování provozu procházejícího filtračním serverem. Díky tomuto rozdělení je možné ve výsledcích testů odlišit jednotlivé provozy. Nakonec byl nastaven rozsah zdrojových portů, které mohl generátor použít pro transportní protokoly, na čísla 57 000 až 58 000. Tím je možné definovat přesný rozsah ve filtračních pravidlech.



Obrázek 5.4: Nastavení zátěže a časového průběhu testu pro nižší rychlost.

5.7.4. netfilter + iptables

Bez filtrace, nižší rychlost

Jak bylo naznačeno v předchozích odstavcích, první testy probíhaly bez filtrace. Na grafech A.1 a A.2 lze vidět, že test proběhl v pořádku. Všech 10 715 202 transakcí bylo úspěšných.

Průměrná rychlost odchozího provoz se držela kolem hodnoty 176,05 Mbit/s a generátor vytvářel packety průměrnou rychlostí 335 385 packetů/s. Tyto a obdobné hodnoty jsou vypočteny z časového úseku, kdy Spirent Avalanche 3100B generoval maximální zátěž (tedy od 30. do 60. sekundy). Stejným způsobem je to myšleno i dále v textu, ale již bez upozornění.

Bez filtrace, vyšší rychlost

Podobně úspěšně jako test při nižší rychlosti dopadl i tento test (viz grafy A.3 a A.4). Z celkových 2 583 184 transakcí nebyla zahozena ani jedna transakce. Průměrná přenosová rychlost činila 774,74 Mbit/s a packetová rychlost 88 370 packetů/s.

Filtrace IP adres, nižší rychlost

Pro filtraci na základě zdrojové a cílové adresy byly provedeny tyto příkazy:

```
# iptables -A FORWARD -m physdev --physdev-is-bridged \
    -s 192.168.2.174 -j DROP
# iptables -A FORWARD -m physdev --physdev-is-bridged \
    -s 192.168.2.175 -d 192.168.2.143 -j DROP
```

První příkaz vytváří pravidlo pro zahození veškerého provozu se zdrojovou IP adresou `192.168.2.174`. Druhý příkaz zahazuje provoz mířící na `192.168.2.143`. Aby nebyl ovlivněn další provoz, bylo pravidlo doplněno tak, aby filtrovalo jen provoz pocházející z IP adresy `192.168.2.175` (aby mohl být tento provoz identifikován ve výstupu testu).

Z grafu A.5 vyplývá, že filtrace byla úspěšná – rozsah1 byl vyfiltrován celý, z rozsahu2 jen provoz na vybranou IP adresu a z rozsahu3 téměř nic. K určitým zahozením z povolených komunikací došlo, ovšem nejedná se ani o jednotky procent.

Graf A.6 ukazuje docela nízké hodnoty přenosové a packetové rychlosti. Je to z toho důvodu, že Spirent Avalanche 3100B otevře určitý počet komunikací a další neotevírá, dokud nepřijde odpověď nebo nevyprší limitní časovače původních spojení. Při snížení časového limitu ovšem nedokázaly cílové servery tak rychle odpovídat. Z toho důvodu bylo toto nastavení ponecháno na hodnotě, při které cílové servery dokázaly doručit odpověď v daném limitu (zkoušeno při nastavení bez filtrace).

Packetová rychlost dosáhla maximálně 117 066 packetů/s (průměrně 112 537 packetů/s) a přenosová rychlost maximálně 62,19 Mbit/s (průměrně 59,86 Mbit/s). Vyzkoušeno bylo 3 925 696 transakcí.

Filtrace IP adres, vyšší rychlost

Nastavení filtrace bylo provedeno stejně jako v předešlém případě. Filtrace opět proběhla převážně správně – opět došlo k menšímu počtu nechtěných zahození (viz graf A.7).

Graf A.8, znázorňuje průběh testu z pohledu datové propustnosti a rychlosti generovaných packetů. V tomto případě dosáhla přenosová rychlost maximálně 454,49 Mbit/s, ovšem průměrně to bylo jen 321,21 Mbit/s. Podobně s packetovou rychlostí – maximum činilo 52 537 packetů/s, průměr pak 40 187 packetů/s.

Filtrace UDP portů, nižší rychlost

Byly použity tyto filtrační příkazy:

```
# iptables -A FORWARD -m physdev --physdev-is-bridged \
    -s 192.168.2.174 -p udp -m multiport --sports 57000:58000 -j DROP
# iptables -A FORWARD -m physdev --physdev-is-bridged \
    -s 192.168.2.175 -d 192.168.2.143 -p udp -m udp --dport 53 -j DROP
```

První příkaz využívá modulu `multiport` k definování rozsahu zdrojových portů UDP. Druhý příkaz používá standardní modul `udp` k filtraci jednoho cílového portu. Graf A.9 dokazuje, že byly odfiltrovány správné komunikace, ovšem s docela masivní ztrátou HTTP komunikací (až 18 % pro jednu komunikaci). Bylo celkem vygenerováno 4 053 272 transakcí.

Graf A.10 ukazuje, že oproti filtraci IP adres se mírně zvedla packetová rychlost (průměrně 119 810 packetů/s) a zvedla se i přenosová rychlost – na průměrných cca 66,03 Mbit/s.

Filtrace UDP portů, vyšší rychlost

Nastavení filtračních pravidel opět zůstalo (což zobrazuje graf A.11), jen se změnila zátěž (což dokazuje graf A.12).

Při tomto testu bylo provedeno celkem 2 503 070 transakcí. Přenosová rychlost se držela průměrně kolem 645,77 Mbit/s, rychlost odchozích packetů průměrně kolem 74 266 packetů/s.

Filtrace TCP/UDP, nižší rychlost, bezstavová

Nastavení filtrovacích pravidel proběhlo pomocí následujících příkazů:

```
1  # for I in {1..100}; do
2  #       iptables -A FORWARD -m physdev --physdev-is-bridged \
3            -s 192.168.3.$I -j DROP
4  # done
5  # iptables -A FORWARD -m physdev --physdev-is-bridged -s 192.168.2.174\
6      -d 192.168.2.143 -p tcp -j DROP
7  # iptables -A FORWARD -m physdev --physdev-is-bridged -s 192.168.2.175\
8      -d 192.168.2.153 -p udp -j DROP
```

Počáteční cyklus `for` slouží k vygenerování 100 bezvýznamných filtrovacích pravidel, které slouží jen k většímu zaplnění filtrovací tabulky, a tudíž ke snížení efektivity filtrace (kvůli otestování a porovnání stavové filtrace, která bude následovat). Další příkazy přidávají pravidla pro filtraci TCP a UDP. Nevyužívají se žádné speciální moduly, jen volba `-p`, která používá informace z IP hlavičky.

Graf A.13 ukazuje, že provoz byl správně odfiltrován (avšak částečně i s něčím navíc). Závislost packetové a datové rychlosti na čase je v grafu A.14. Celkem bylo vyzkoušeno 3 342 122 transakcí. Průměrná přenosová rychlost byla 52,02 Mbit/s, průměrná packetová rychlost 95 837 packetů/s. Neúspěšné transakce dosáhly k 17,80 %.

Filtrace TCP/UDP, vyšší rychlost, bezstavová

Stejně jako v předchozích měřeních nastavení filtrace zůstalo, jen se změnila rychlost generovaných dat. V tomto případě se provedlo celkem 2 197 419 transakcí, přenosová rychlost byla vyšší – průměrně 558,03 Mbit/s a packetová rychlost nižší – průměrně 65 168 packetů/s (viz také grafy A.15 a A.16). Z celkového počtu transakcí bylo provedeno 8,49 % neúspěšných transakcí.

Filtrace TCP/UDP, nižší rychlost, stavová

Aby mohla být vůbec filtrace s hlídáním stavu vyzkoušena, muselo být nejdříve zajištěno to, aby filtračním serverem procházel veškerý provoz (tedy i odpovědi od cílových serverů ke generátoru provozu). Tento požadavek vyplývá z logiky stavové filtrace – aby si mohl firewall zaznamenávat stav dané komunikace, musí jím procházet oba směry.

Přepínač Mikrotik-OUT z obrázku 5.3 naštěstí umožňoval vložit pravidla, kterými lze provoz přesměrovávat mezi porty. Stačilo tedy přidat pravidla, aby vše, co přichází z IP adres `192.168.2.143` a `192.168.2.153`, posílal portem na filtrační server místo optického propoje na Mikrotik-IN.

Při zkoušení stavové filtrace byl proveden tento příkaz:

```
1  # iptables -A FORWARD -m physdev --physdev-is-bridged -m state \
2      --state ESTABLISHED,RELATED -j ACCEPT
```

Za ním následovaly stejné `iptables` příkazy, jako při testování bezstavové filtrace, jen doplněné o parametry: `-m state --state NEW`.

Graf A.17 zobrazuje, že došlo k víceméně správnému zahození provozu. Na grafu A.18 to sice není příliš vidět, ale došlo i k mírnému zlepšení oproti bezstavovému testu. Čísla to dokazují: bylo uskutečněno 3 604 203 transakcí při průměrné přenosové rychlosti 56,47 Mbit/s a průměrné packetové rychlosti 102 839 packetů/s. Tedy došlo k více transakcím a ubylo transakcí neúspěšných, kterých bylo 17,59 %.

Filtrace TCP/UDP, vyšší rychlost, stavová

Stavová filtrace při vyšší přenosové rychlosti je prakticky ten samý případ jako předchozí: stavová filtrace dosáhla o něco málo lepších výsledků než filtrace bezstavová, ovšem grafy A.19 a A.20 to příliš nedokazují. Celkem proběhlo 2 471 982 transakcí, z toho neúspěšných bylo 3,66 %. Průměrná přenosová rychlost byla 641,74 Mbit/s, průměrná packetová rychlost byla 73 541 packetů/s.

Pro další důkaz, že se hlídání stavu aplikovalo, je možné nahlédnout do výpisu pravidel `iptables`. Čítače dokazují, že dané stavové pravidlo bylo opravdu použito (výpis byl zkrácen):

```
1  Chain FORWARD (policy ACCEPT 587K packets, 288M bytes)
2   pkts bytes target     prot opt   source          destination
3  2204K  800M ACCEPT     all  --    0.0.0.0/0       0.0.0.0/0    \
4                                                  state RELATED,ESTABLISHED
5      0     0 DROP       all  --    192.168.3.1     0.0.0.0/0
6  ...
7      0     0 DROP       all  --    192.168.3.100   0.0.0.0/0
8   4056  178K DROP       tcp  --    192.168.2.174   192.168.2.143
9   6085  335K DROP       udp  --    192.168.2.175   192.168.2.153
```

Z uvedeného výpisu je vidět, že první pravidlo (na řádku číslo 3 a 4) akceptující již jednou povolená spojení, bylo aplikováno na 2,204 milionů packetů. Lze tedy prohlásit, že transparentní stavová filtrace v netfilteru a `iptables` skutečně funguje.

5.7.5. nftables

Bez filtrace, nižší rychlost

I v tomto případě byl proveden nejdříve test bez filtrace. Grafy A.21 a A.22 ukazují podobné hodnoty jako netfilter a `iptables` (i když došlo k mírným ztrátám). Průměrná rychlost dat ukazovala 176,03 Mbit/s, průměrná rychlost odesílání packetů byla 335 344 packetů/s. Vyzkoušelo se celkem 10 715 202 transakcí.

Bez filtrace, vyšší rychlost

Grafy A.23 a A.24 se opět oproti netfilteru s `iptables` příliš nezměnily. A samotná čísla také ne: celkem 2 583 184 transakcí, přenosová rychlost průměrně 774,78 Mbit/s, packetová rychlost v průměru 88 371 packetů/s.

Filtrace IP adres, nižší rychlost

Tento test byl proveden při konfiguraci filtrace pomocí příkazů:

```
# nft add rule bridge filter forward ether type ip \
    ip saddr 192.168.2.174 drop
# nft add rule bridge filter forward ether type ip \
    ip saddr 192.168.2.175 ip daddr 192.168.2.143 drop
```

První příkaz filtruje na základě zdrojové IP adresy (`ip saddr`), druhý příkaz na základě zdrojové i cílové IP adresy (`ip daddr`).

Z grafu A.25 lze vyčíst, že filtrace proběhla správně. Graf A.26 ukazuje nízkou přenosovou rychlost i packetovou rychlost, ovšem vyšší než předchozí zkoušený nástroj.

Celkem proběhlo 4 203 499 transakcí při průměrné datové propustnosti 68,57 Mbit/s a rychlosti generování packetů 129 562 packetů/s. Všechna tato čísla jsou lepší, než čeho dosáhl netfilter.

Filtrace IP adres, vyšší rychlost

Nastavení zůstalo stejné, jako v minulém případě. Jen se změnily statistické údaje a graf A.28. Jelikož se filtrovalo to samé a k žádnému většímu nechtěnému zahození (které by se výrazně projevilo) nedošlo, graf A.27 zůstal téměř nezměněn.

Při průměrné přenosové rychlosti 321,67 Mbit/s a packetové rychlosti 40 252 packetů/s bylo provedeno 1 223 915 transakcí. Tato čísla jsou srovnatelná s netfilterem.

Filtrace UDP portů, nižší rychlost

Pro filtrování UDP portů byly provedeny tyto příkazy:

```
# nft add rule bridge filter forward ether type ip \
    ip saddr 192.168.2.174 udp sport 57000-58000 drop
# nft add rule bridge filter forward ether type ip \
    ip saddr 192.168.2.175 ip daddr 192.168.2.143 udp dport 53 drop
```

První příkaz filtruje UDP provoz jen z dané IP adresy a z rozsahu zdrojových portů `57000-58000`. Druhý příkaz filtruje pro vybranou dvojici IP adres (zdroj–cíl) jen cílový UDP port 53 (tedy standardně DNS provoz).

Při tomto testu bylo dosaženo průměrné rychlosti generování packetů 334 264 packetů/s a průměrné přenosové rychlosti 175,63 Mbit/s. Proběhlo dohromady 10 714 896 transakcí. Na rozdíl od netfilteru s `iptables`, nedošlo při tomto testu k žádnému nechtěnému zahození (viz graf A.29) a bylo dosaženo vyšší přenosové i packetové rychlosti, což lze vidět v grafu A.30.

Filtrace UDP portů, vyšší rychlost

Srovnáme-li opět s netfilterem a `iptables`, zjistíme, že při tomto testu byly nftables opět výkonnější. Počet transakcí se příliš nezměnil (dosáhl čísla 2 583 320), ovšem přenosová rychlost se zvýšila na průměrnou hodnotu 768,01 Mbit/s a packetová rychlost porostla na průměrnou hodnotu 88 766 packetů/s. Grafy A.31 a A.32 výsledek jen podtrhují.

Filtrace TCP/UDP, nižší rychlost

Jelikož dle [21] není aktuálně v nftables podpora pro transparentní stavovou filtraci implementována, nelze ji vyzkoušet. Proto byla vyzkoušena jen bezstavová filtrace.

Podobně jako v předchozích testech, byla i v tomto měření vygenerována bezvýznamná výplňová pravidla:

```
1  # for I in {1..100}; do
2  #       nft add rule bridge filter forward ether type ip \
3            ip saddr 192.168.3.$I drop
4  # done
5  # nft add rule bridge filter forward ether type ip \
6      ip saddr 192.168.2.174 ip daddr 192.168.2.143 ip protocol tcp drop
7  # nft add rule bridge filter forward ether type ip \
8      ip saddr 192.168.2.175 ip daddr 192.168.2.153 ip protocol udp drop
```

Cyklus `for` tedy opět vytvoří pravidla, která se na provoz neaplikují. Druhý `nft` příkaz zakáže propuštění TCP provozu pro vybranou dvojici IP adres. Poslední příkaz zakáže pro danou dvojici IP adres provoz UDP.

Graf A.33 ukazuje, že kromě správného provozu byl ve velké míře odfiltrován i nesprávný provoz. Graf A.34 zase ukazuje rychlost přenosu dat, která byla průměrně 48,20 Mbit/s, a rychlost generování packetů, která dosáhla průměru 86 450 packetů/s. Celkem bylo provedeno 2 935 934 transakcí. Uvedené hodnoty jsou horší než u netfilteru. Jde vidět, že nftables má problémy s více pravidly. V tomto případě by bylo efektivnější použít rozsah adres v jednom pravidle než 100 samostatných pravidel.

Filtrace TCP/UDP, vyšší rychlost

Opět horší výsledek proti stejnému měření u netfilteru + `iptables` (co se týká úspěšnosti přenosu dat, viz graf A.35). Na grafu A.36 lze vidět, že kolem 36. sekundy se odehrál nějaký problém, který se zřejmě podepsal na vysoké ztrátovosti HTTP komunikace na cílový server `192.168.2.153`. Nejspíš nějaký zádrhel na cílovém serveru, jelikož další průběh na problémy s filtračním serverem neukazuje. Zajímavé ovšem je, že i přesto bylo dosaženo vyšší přenosové rychlosti než u netfilteru.

Celkově bylo provedeno 2 350 233 transakcí při průměrné přenosové rychlosti 594,22 Mbit/s a průměrné packetové rychlosti 66 865 packetů/s.

5.7.6. PF_RING

Bez filtrace, nižší rychlost

Jako v předchozích dvou případech, i PF_RING byl nejdříve otestován při dvou rychlostech datové propustnosti bez filtrace. Nejdříve nižší přenosová rychlost, ovšem vyšší packetová

rychlost. Jak lze vidět na grafu A.37, došlo k nechtěným zahozením HTTP i DNS provozu. Graf A.38 se také odlišuje od stejných grafů předchozích nástrojů – je vidět, že s touto packetovou rychlostí měl PF_RING problémy. To se kromě packetové rychlosti, která dosáhla průměru 162 180 packetů/s, projevilo i na průměrné přenosové rychlosti – 87,60 Mbit/s, a samozřejmě také na celkovém počtu vyzkoušených transakcí – 5 720 014. Celkem bylo neúspěšných cca 10,08 % transakcí.

Bez filtrace, vyšší rychlost

Při vyšší přenosové rychlosti (a s většími packety) bylo neúspěšných jen 981 transakcí (z celkového počtu 2 582 916 transakcí). V relativních jednotkách se jedná o 0,04 %, takže v tomto testu se PF_RING dá srovnávat s předchozími nástroji. I když čísla jsou menší než například u nftables: průměrná datová propustnost byla 724,28 Mbit/s, průměrná rychlost generování packetů byla 82 792 packetů/s. Viz také grafy A.39 a A.40.

Filtrace IP adres, nižší rychlost

Filtrace podle IP adres byla provedena těmito příkazy:

```
1  (term1)# ./pfbridge -a eth1 -b eth3 -f "not src host 192.168.2.174 and\
2                 not (src host 192.168.2.175 and dst host 192.168.2.143)"
3  (term2)# ./pfbridge -a eth3 -b eth1 -f "not src host 192.168.2.174 and\
4                 not (src host 192.168.2.175 and dst host 192.168.2.143)"
```

Stejně jako při předchozích měřeních, byly spuštěny dvě instance `pfbridge` se stejnou filtrovací funkcí. Zde je pomocí BPF syntaxe vyfiltrována nejdříve jedna zdrojová IP adresa (`not src host 192.168.2.174`) a následně dvojice zdrojová–cílová IP adresa (`not (src host 192.168.2.175 and dst host 192.168.2.143)`).

Důkaz, že filtrace proběhla správně, podává graf A.41. Graf A.42 ukazuje průběhy packetové a bitové rychlosti, které jsou srovnatelné se stejným testem u nftables. Bylo provedeno celkem 4 202 463 transakcí, při průměrné packetové rychlosti 129 392 packetů/s a průměrné bitové rychlosti 68,48 Mbit/s.

Filtrace IP adres, vyšší rychlost

Nastavení opět zůstalo. Výsledek testu lze srovnávat s nftables (viz grafy A.43 a A.44). Ačkoliv průměrná čísla jsou mírně horší: celkem bylo vyzkoušeno 1 217 705 transakcí. Přenosová rychlost dosáhla průměru 311,70 Mbit/s (maximálně pak 401,43 Mbit/s), packetová rychlost průměru 39 529 packetů/s (maximálně 49 180 packetů/s).

Filtrace UDP portů, nižší rychlost

Pro vyfiltrování části UDP provozu byly použity tyto příkazy:

```
1  (term1)# ./pfbridge -a eth1 -b eth3 -f "not (src host 192.168.2.174 \
2                 and udp src portrange 57000-58000) \
3                 and not (src host 192.168.2.175 \
4                 and dst host 192.168.2.143 and udp dst port 53)"
5  (term2)# ./pfbridge -a eth3 -b eth1 -f "not (src host 192.168.2.174 \
```

```
6                   and udp src portrange 57000-58000) \
7                   and not (src host 192.168.2.175 \
8                   and dst host 192.168.2.143 and udp dst port 53)"
```

Příkazy provedou to, že z adresy `192.168.2.174` vyfiltrují všechny zdrojové UDP porty z rozsahu 57 000–58 000 a zakážou také komunikaci od zařízení `192.168.2.175` k zařízení `192.168.2.143` na cílovém portu 53.

Že příkazy zřejmě dělají, co mají, ale PF_RING vykazoval problémy, lze vidět na grafu A.45. Je ale patrné, že nedošlo k úplnému vyfiltrování DNS komunikace na server `192.168.2.143`. Celkově vzato měl PF_RING s touto filtrací problémy. Graf A.46 ukazuje časový průběh testu z pohledu přenosové rychlosti a packetové rychlosti, který ovšem příliš nekoresponduje s grafy stejných měření ostatních nástrojů. Průměrná přenosová rychlost byla 58,25 Mbit/s a průměrná packetová rychlost byla 106 302 packetů/s. Celkem bylo provedeno 3 953 903 transakcí, což je proti nftables daleko horší výsledek.

Filtrace UDP portů, vyšší rychlost

Výsledky testu pro vyšší přenosovou rychlost jsou opět horší než u nftables. Graf A.47 ukazuje docela velký počet nechtěných ztrát. Graf průběhu měření A.48 je již více podobný obdobným grafům u předchozích nástrojů, ovšem podstatný rozdíl tady je. Po 60. sekundě je vidět strmý pád obou rychlostí. To bylo způsobeno tím, že v jádře operačního systému došlo k chybě, která vyvolala tzv. kernel panic. Ten se projevoval i při opakovaných pokusech.

Celkem proběhlo 2 155 390 transakcí při průměrné rychlosti přenosu dat 700,26 Mbit/s a průměrné packetové rychlosti 81 249 packetů/s.

Filtrace TCP/UDP, nižší rychlost

Jak je možné z předchozího textu vytušit, ani PF_RING nenabízí možnost stavové filtrace (jedná se o jednoúčelovou utilitu, která přeposílá packety z jednoho síťového rozhraní na druhé a při tom umožňuje nastavit pravidla, který provoz může projít a který ne). Proto ani v tomto případě nebude stavová filtrace vyzkoušena.

Pro vyfiltrování jednoho sta zdrojových IP adres a TCP a UDP provozu pro určité zdrojové a cílové adresy, byly použity příkazy, které je možné nalézt v příloze A.5. Kromě bezpočtu „pravidel“ pro filtraci zdrojových IP adres, je na konci část, která filtruje TCP a UDP komunikace pro vybrané dvojice IP adres:

```
1     ...
2     and not (src host 192.168.2.174 and dst host 192.168.2.143 \
3     and ip proto \tcp) and not (src host 192.168.2.175 \
4     and dst host 192.168.2.153 and ip proto \udp)"
```

Podobně jako nftables, i PF_RING v tomto testu příliš neobstál (vykazuje ještě horší výsledky než nftables): v tomto případě bylo provedeno celkem 2 654 032 transakcí. Průměr přenosové rychlosti byl 44,03 Mbit/s a průměr packetové rychlosti byl 78 599 packetů/s (viz také graf A.50). Úspěšnost filtrace ukazuje graf A.49.

Filtrace TCP/UDP, vyšší rychlost

Nastavení zůstalo stejné jako v předchozím případě. Zcela nepochopitelně došlo dle grafu A.51 k úplné neúspěšnosti HTTP komunikací na cílový server `192.168.2.153`, ačkoliv tato filtrace nebyla nastavena.

Průměrná přenosová rychlost byla okolo 415,33 Mbit/s. Průměrná packetová rychlost pak byla 47 860 packetů/s (viz také graf A.52). Celkem bylo provedeno 1 719 504 transakcí. Každopádně i zde je výsledek horší než u nftables.

5.7.7. netmap s vlastními ovladači NIC

Bez filtrace, nižší rychlost

Na počátku testování netmapu byl opět proveden test bez filtrace. Výsledky nepřekvapí, opět jsou víceméně stejné jako u předchozích nástrojů (s výjimkou PF_RINGu). Celkem provedeno 10 716 426 transakcí. Žádná transakce nebyla neúspěšná (viz také graf A.53). Graf A.54 ukazuje přenosovou a packetovou rychlost. Průměr přenosové rychlosti dosáhl 176,16 Mbit/s, průměr packetové rychlosti dosáhl 335 589 packetů/s.

Bez filtrace, vyšší rychlost

V tomto testu také nedošlo k zahození žádné transakce, proto je graf A.55 celkem nezajímavý. Průměrná packetová rychlost byla 88 425 packetů/s a průměrná přenosová rychlost se držela kolem 775,23 Mbit/s (viz také graf A.56). Bylo vyzkoušeno celkem 2 583 184 transakcí.

Filtrace IP adres, nižší rychlost

Pro přidání pravidel pro filtraci IP adres byly použity příkazy:

```
1   # ./ipfw add drop ip from 192.168.2.174 to any
2   # ./ipfw add drop ip from 192.168.2.175 to 192.168.2.143
```

První příkaz zakáže veškerou IP komunikaci z dané adresy (`from 192.168.2.174`), druhý příkaz zakáže všechnu IP komunikaci z vybrané adresy putující na cílový server (`to 192.168.2.143`).

Stejně jako v předchozích případech, i zde nebylo dosaženo vysoké přenosové rychlosti (průměrně 68,57 Mbit/s) ani packetové rychlosti (průměrně 129 548 packetů/s). Viz také graf A.58. Dle grafu A.57 ale došlo ke správnému odfiltrování provozu. Provedeno bylo celkem 4 203 514 transakcí.

Filtrace IP adres, vyšší rychlost

I při vyšší rychlosti zůstalo stejné nastavení filtrace (viz graf A.59). Při celkovém počtu 1 224 062 transakcí bylo dosaženo packetové rychlosti průměrně 40 253 packetů/s a průměrné datové propustnosti 321,67 Mbit/s (průběh lze vidět v grafu A.60).

Filtrace UDP portů, nižší rychlost

Pro odfiltrování vybrané části UDP provozu byly použity příkazy:

```
1   # ./ipfw add drop udp from 192.168.2.174 57000-58000 to any
2   # ./ipfw add drop udp from 192.168.2.175 to 192.168.2.143 dst-port 53
```

Za slovem `drop` lze vidět, že vybíráme jen UDP provoz. Rozsah zdrojových portů se vkládá ihned za zdrojovou IP adresu. Cílový port se vkládá za cílovou IP adresu v parametru `dst-port`.

Jak je možné vidět v grafu A.61, byl odfiltrován správný provoz. Graf A.62 ukazuje časový průběh přenosové rychlosti, která byla v průměru 175,75 Mbit/s, a packetové rychlosti, která dosáhla průměru 334 491 packetů/s. Celkový počet transakcí dosáhl hodnoty 10 716 426.

Filtrace UDP portů, vyšší rychlost

Při průměrné přenosové rychlosti 768,38 Mbit/s a průměrné packetové rychlosti 88 809 packetů/s (viz také graf A.64) bylo dosaženo 2 583 184 provedených transakcí. Podle grafu A.63 proběhla filtrace dle předpokladu.

Filtrace TCP/UDP, nižší rychlost, bezstavová

Netmap ve svém firewallu `netmap-ipfw` umožňuje zadat pravidla pracující se stavem spojení, proto může být stavová filtrace vyzkoušena. Nejdříve však proběhnou testy bez stavových pravidel.

```
1   # for I in {1..100}; do
2   #       ./ipfw add drop ip from 192.168.3.$I to any
3   # done
4   # ./ipfw add drop tcp from 192.168.2.174 to 192.168.2.143
5   # ./ipfw add drop udp from 192.168.2.175 to 192.168.2.153
```

Cyklus `for` opět přidá 100 jednoduchých pravidel, které se ale v testu neaplikují. Další `ipfw` příkaz vyfiltruje TCP komunikaci mezi dvěma stanicemi a poslední příkaz vyfiltruje veškerou UDP komunikaci mezi jinými dvěma stanicemi.

Správnost filtrace dokazuje graf A.65. Při tomto testu se průměrná přenosová rychlost držela kolem 138,63 Mbit/s a průměrná rychlost generování packetů kolem 263 736 packetů/s (průběhy lze vidět v grafu A.66). Celkově bylo vygenerováno 9 495 123 transakcí.

Filtrace TCP/UDP, vyšší rychlost, bezstavová

Při vyšší přenosové rychlosti bylo dosaženo průměrně 744,30 Mbit/s (viz také graf A.68). Packetová rychlost dosáhla průměru 84 418 packetů/s. Že nastavení filtrace zůstalo stejné jako v předchozím případě a že nebylo zahozeno nic navíc, zobrazuje graf A.67. Celkem bylo provedeno 2 579 755 transakcí.

Filtrace TCP/UDP, nižší rychlost, stavová

Stejně jako u netfilteru a `iptables` bylo v tomto případě na přepínačích připojených k filtračnímu serveru nastaveno posílání veškerého provozu přes testovací server.

Filtrování bylo provedeno tímto způsobem:

```
1  # ./ipfw add check-state
2  # for I in {1..100}; do
3  #       ./ipfw add drop ip from 192.168.3.$I to any
4  # done
5  # ./ipfw add drop tcp from 192.168.2.174 to 192.168.2.143
6  # ./ipfw add drop udp from 192.168.2.175 to 192.168.2.153
7  # ./ipfw add allow tcp from any to any setup keep-state
8  # ./ipfw add allow udp from any to any setup keep-state
```

Příkaz na řádku 1 vloží jako první pravidlo akci `check-state`, která se stará o zkontrolování komunikace vůči seznamu již povolených spojení a jeho případné povolení. Dále následuje přidání jednoho sta bezvýznamných pravidel. Následuje zakázání TCP a UDP komunikace pro vybrané dvojice IP adres (stejně jako při bezstavovém filtrování). A poslední dva příkazy povolují všechny TCP a UDP komunikace, a zároveň by povolenou komunikaci měly zavést do seznamu povolených spojení (`setup keep-state`).

Dle grafu A.69 došlo k odfiltrování správné komunikace, ovšem při tom byl nepatrně zahozen i provoz, který měl zůstat povolen (HTTP provoz z rozsahu2 a rozsahu3 na `192.168.2.153`). Graf A.70 zobrazuje časovou závislost přenosové a packetové rychlosti, které v tomto případě dosáhly průměrných hodnot 92,97 Mbit/s a 174 608 packetů/s. Celkem se vygenerovalo 6 197 999 transakcí.

Jak je z výsledků a grafů vidět, na rozdíl od netfilteru, nebylo proti bezstavovému testu dosaženo lepšího výsledku (došlo dokonce k mírnému zhoršení kvůli zahození i jiného provozu, než který byl vyžadován).

Filtrace TCP/UDP, vyšší rychlost, stavová

Test pro vyšší rychlost je podobný stavu pro nižší rychlost: i zde bylo dosaženo spíše horších čísel než u bezstavové filtrace (ačkoliv ne o moc). Průměrná přenosová rychlost byla 740,74 Mbit/s, průměrná packetová rychlost byla 83 931 packetů/s (viz také graf A.72). I při tomto testu bylo zahozeno určité množství nechtěného provozu (dle grafu A.71). Počet provedených transakcí dosáhl počtu 2 575 755.

V tomto případě může vyvstat otázka, zda-li se pravidla pro kontrolu stavu a povolení spojení (případně zavedení do seznamu spojení) vůbec aplikují. Zde to opravdu vypadá, že k hlídání stavu skutečně nedojde. Lze tak usoudit na základě výpisu pravidel s čítači jejich použití (výpis je zkrácen):

```
1  00100        0          0 check-state
2  00200        0          0 deny ip from 192.168.3.1 to any
3  ...
4  10100        0          0 deny ip from 192.168.3.100 to any
5  10200     4061     178640 deny tcp from 192.168.2.174 to 192.168.2.143
6  10300     6078     334290 deny udp from 192.168.2.175 to 192.168.2.153
```

```
7   10400   140954   33085789 allow tcp from any to any setup keep-state
8   10500        0          0 allow udp from any to any setup keep-state
9   65535 40744522 2852146698 allow ip from any to any
```

Číslo v prvním sloupci označuje pořadové číslo pravidla. Druhé číslo označuje počet packetů, na které bylo pravidlo aplikováno. Třetí označuje počet bajtů, které daným pravidlem „prošly“. Zbytek řádku ukazuje samotné pravidlo firewallu.

Jak lze vidět, pravidlo pro zkontrolování spojení a jeho případné povolení, nebylo použito vůbec. Stejně tak kupodivu nebylo aplikováno pravidlo pro povolení UDP komunikací a jejich zavedení do seznamu povolených spojení, ačkoliv stejné pravidlo pro TCP spojení aplikováno bylo.

S největší pravděpodobností tedy netmap umožňuje vložit pravidla pro stavovou filtraci, ovšem samotné hlídání stavu není ve skutečnosti implementováno. Jak bylo uvedeno v sekci 3.3.4, je netmap původně vyvinut pro operační systém FreeBSD. Je tedy docela možné, že stavová filtrace nefunguje jen Linuxu, ovšem ve FreeBSD ano.

5.8. Agregace síťových rozhraní v experimentální síti

Jak již bylo v předchozích odstavcích naznačeno, díky univerzálnosti síťové infrastruktury experimentální sítě a také filtračního serveru, bylo možné jej zapojit tak, aby mohla být vyzkoušena agregace síťových linek. Původní dvouportová síťová karta Intel byla vyměněna za čtyřportovou verzi. Dvě rozhraní pak byla připojena k přepínači Mikrotik-IN a dvě rozhraní k přepínači Mikrotik-OUT. Dohromady tedy vytvořily dvě linky o přenosové kapacitě 2 Gbit/s. Zapojení ukáže názorněji obrázek 5.5.

Při zkoumání podpory vybraných filtračních nástrojů bylo zjištěno, že ani PF_RING ani netmap neumožňují seskupování síťových rozhraní. Jedinými zástupci pro tento test tedy byly netfilter + `iptables` a nftables, jelikož využívají standardní linuxový síťový zásobník, který tuto funkci podporuje.

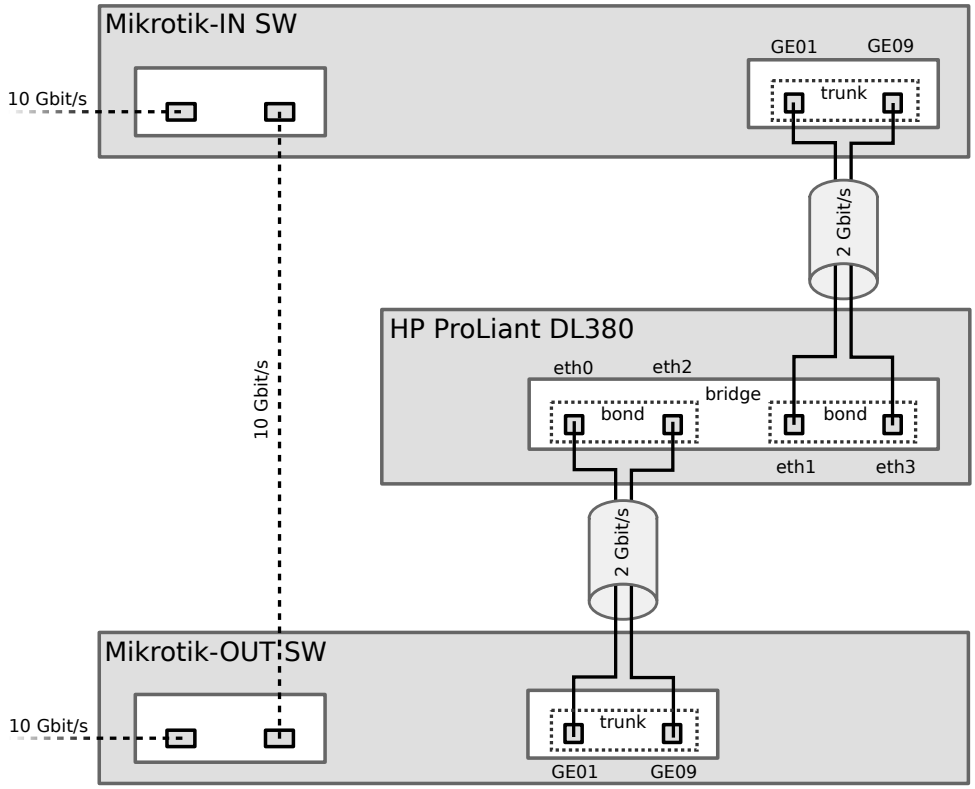
5.8.1. Nastavení bondingu

Dle kapitoly 4 lze v Linuxu pro agregaci linek použít dvě řešení – NIC bonding a NIC teaming. V tomto případě byl vybrán NIC bonding, jelikož bývá standardní součástí linuxového jádra, a není tedy nutné nic doinstalovávat.

Nejdříve je potřeba vytvořit virtuální bonding rozhraní. To můžeme provést tímto způsobem:

```
1   # ip link add name bond0 type bond mode balance-xor \
2         xmit_hash_policy layer3+4
3   # ip link add name bond1 type bond mode balance-xor \
4         xmit_hash_policy layer3+4
```

Těmito příkazy vytvoříme dvě rozhraní `bond0` a `bond1` typu `bond`. Jako mód pro vyvažování zátěže je zvolen `balance-xor`, který rozprostírá zátěž tak, že pro každý packet vypočítá z informací v jeho hlavičkách hodnotu, jenž funguje jako identifikátor odchozího fyzického rozhraní. Jak samotný název módu napovídá, pro výpočet hodnoty se používá operace XOR a také modulo. Ve výchozím stavu se hodnota počítá jako [(zdrojová



Obrázek 5.5: Tímto způsobem byl filtrační server připojen k přepínačům. Čárkovaná čára znamená optické propojení, plná čára značí metalické propojení.

MAC adresa XOR cílová MAC adresa XOR ID typu packetu) modulo počet podřízených rozhraní]. V tomto případě je však hned následujícím parametrem `xmit_hash_policy` výpočet upraven tak, aby se hodnota počítala z hlaviček síťové a transportní vrstvy ISO OSI modelu. Více v [22]. Uvedený mód byl vybrán tak, aby co nejvíce odpovídal módu vyvažování na straně přepínačů.

Následně je potřeba přiřadit jednotlivé fyzické porty k bonding rozhraním a tato rozhraní aktivovat:

```
1   # ip link set eth1 master bond0
2   # ip link set eth3 master bond0
3   # ip link set eth0 master bond1
4   # ip link set eth2 master bond1
5   # ip link set bond0 up
6   # ip link set bond1 up
```

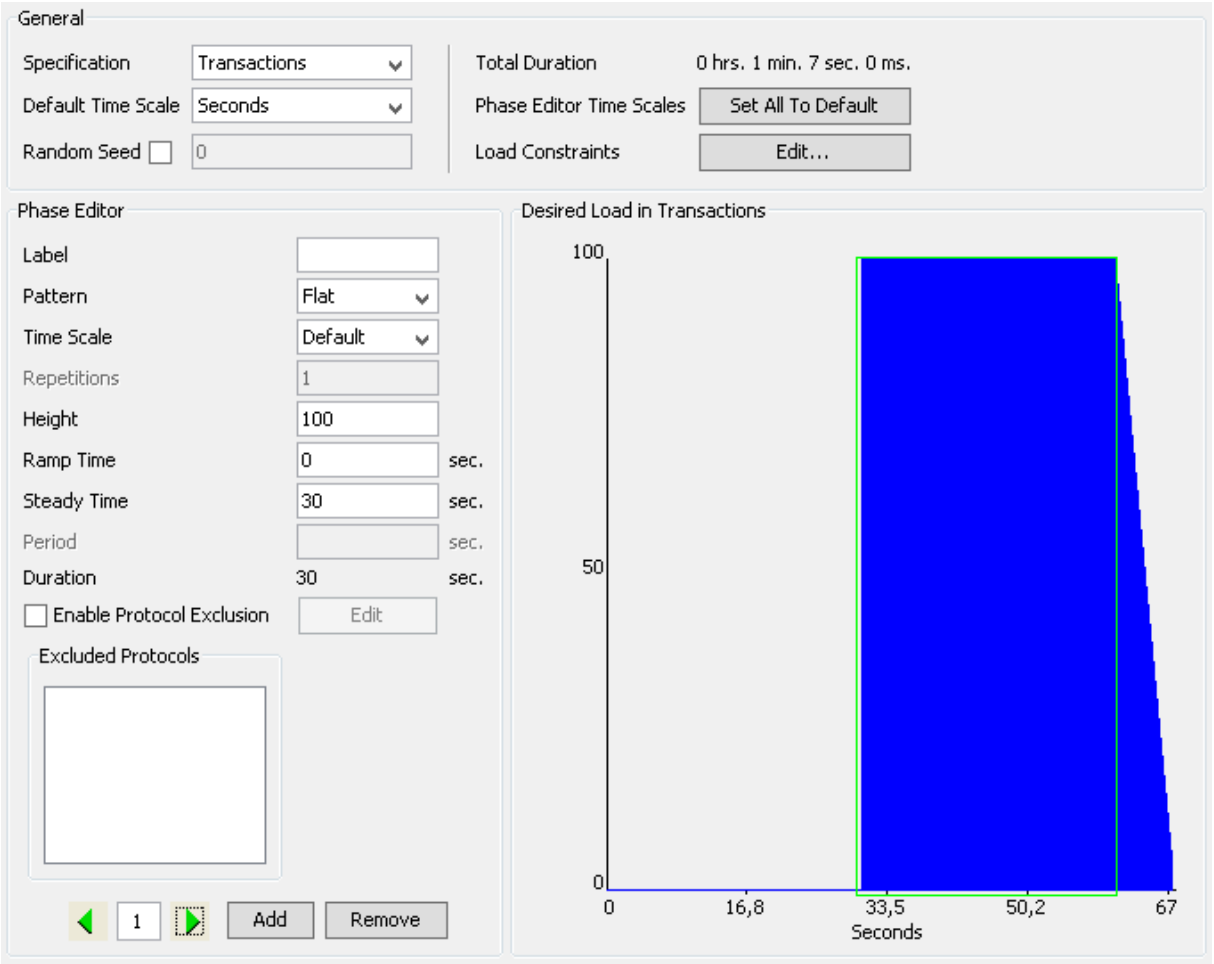
Nakonec se vytvoří virtuální rozhraní pro síťový most, do kterého se zařadí obě bonding rozhraní:

```
1   # ip link add br0 type bridge
2   # ip link set bond0 master br0
3   # ip link set bond1 master br0
4   # ip link set br0 up
```

Příkazem na řádku číslo 4 se síťový most aktivuje a server by měl začít přeposílat provoz z jednoho bonding rozhraní na druhé, a to přenosovou rychlostí až 2 Gbit/s.

5.8.2. Nastavení Spirent Avalanche 3100B

Generovaná zátěž byla nastavena ve velké míře stejně, jako zátěž pro vyšší rychlost z předchozího testování (viz podsekci 5.7.3). Jediný rozdíl byl ten, že místo 1,4kB souboru se v HTTP POST požadavcích posílal soubor velký 1 MB. Druhou změnou bylo použití jiné specifikace zátěže. Místo „SimUsers/second“ (tedy simulovaných uživatelů za sekundu) byla zátěž definována v transakcích (viz obrázek 5.6). Díky této změně generoval Spirent Avalanche 3100B stabilnější zátěž než v případě použití simulovaných uživatelů. Jak je na obrázku vidět, časový průběh zůstal stejný.



Obrázek 5.6: Nastavení zátěže a časového průběhu testu při testování linkové agregace.

Při testování filtrace na agregovaných linkách byla zkoušena jen jedna rychlost, nastavená dle předchozího odstavce. Testovat na nižší rychlosti nedávalo smysl, jelikož zkoušíme přenosovou rychlost nad 1 Gbit/s. Výše popsaným způsobem byl vygenerován tok o velikosti v průměru 1,4 Gbit/s, maximálně pak přibližně 1,6 Gbit/s. Packetová rychlost dosáhla průměru zhruba 116 tisíc packetů za sekundu. Rozložení velikosti packetů ve vygenerovaném datovém toku je v tabulce 5.4.

Jelikož byly do experimentální sítě zapojeny jen gigabitová rozhraní z Avalanche, byl pro tento test alokován další port (konkrétně port číslo 3), takže mohl být vygenerován síťový provoz o velikosti větší než 1 Gbit/s. Adresní prostor, který byl k dispozici, musel

Velikost packetu (bajty)	Procentuální zastoupení (%)
0–64	1,00
65–128	0,33
1 025–1 500	8,33
1 500+	90,33

Tabulka 5.4: Rozložení velikostí packetů v datovém toku při testování bondingu (cca 1,4 Gbit/s). Protože bylo nastaveno posílání souboru, jehož obsah se do jednoho packetu nevejde, byly Avalanchem HTTP dotazy fragmentovány do packetů, jejichž velikost na linkové vrstvě ISO OSI modelu dosahovala 1 514 bajtů (resp. 1 518 B s kontrolním součtem).

být rozdělen mezi porty tak, aby se nepřekrýval. Přiřazení IP adres k portům a jejich rozdělení bylo provedeno tímto způsobem:

- `192.168.2.174`, port č. 4 (dále označen jako rozsah1),
- `192.168.2.175`, port č. 4 (dále označen jako rozsah2),
- `192.168.2.176`, port č. 4 (dále označen jako rozsah3),
- `192.168.2.177-192.168.2.179`, port č. 3 (dále označen jako rozsah4).

To jsou veškeré úpravy, které byly oproti nastavení z předchozího testování provedeny. Nastavení filtračních nástrojů v jednotlivých testech, a případná další nastavení, byla provedena stejně jako v sekci 5.7.

5.8.3. netfilter + iptables

Bez filtrace

Jako první byl opět testován netfilter s `iptables`. Nejdříve bez nastavení jakékoliv filtrace.

Při této zkoušce bylo vygenerováno 375 031 transakcí, z toho 249 transakcí neúspěšných, což znamená 99,93% úspěšnost (viz také graf A.73). Bylo dosaženo maximální přenosové rychlosti 1,59 Gbit/s (průměrná rychlost 1,38 Gbit/s) a průměrné packetové rychlosti 116 531 packetů/s (průběh je v grafu A.74).

Z výsledků a grafů vyplývá, že při použití bondingu dochází v malé míře k zahazování legitimního provozu.

Filtrace IP adres

V tomto případě bylo provedeno celkem 270 793 transakcí při průměrné rychlosti generování packetů 83 139 packetů/s a průměrné datové rychlosti 981,71 Mbit/s. Více také v grafech A.75 a A.76.

I v tomto případě došlo zahození provozu, který měl zůstat povolen. Konkrétně bylo neúspěšných až 6 % HTTP provozu na cílový server `192.168.2.153` z rozsahu2.

Filtrace UDP portů

Při filtraci UDP packetů bylo dosaženo podobné přenosové rychlosti, jako bez filtrace – průměrně 1,37 Gbit/s (většinu provozu tvořil HTTP provoz, který nebyl filtrován). Průměrná packetová rychlost byla 115 976 packetů/s (viz také graf A.78). Celkově bylo provedeno 376 400 transakcí. Graf A.77 ukazuje, že v omezené míře došlo opět k nechtěným zahozeným transakcím.

Filtrace TCP/UDP, bezstavová

Při bezstavové verzi tohoto testu bylo vyzkoušeno celkem 269 019 transakcí. Průměr přenosové rychlosti dosáhl 921,58 Mbit/s (maximum pak 1,02 Gbit/s). Packetová rychlost byla průměrně 78 075 packetů/s. Časové průběhy těchto veličin jsou k dispozici v příloze v grafu A.80.

TCP a UDP provoz byl vyfiltrován správně, ovšem částečně i s provozem navíc (dle grafu A.79).

Filtrace TCP/UDP, stavová

Díky tomu, že netfilter s `iptables` podporuje stavovou filtraci, bylo možné ji i tentokrát vyzkoušet.

Při tomto testu bylo provedeno celkově 300 377 transakcí. A to při průměrné přenosové rychlosti 1,03 Gbit/s (maximální rychlost 1,19 Gbit/s) a průměrné packetové rychlosti okolo 86 904 packetů/s. Závislost těchto veličin na čase lze zhlédnout na grafu A.82. Úspěšnost filtrace lze dohledat v grafu A.81.

Ačkoliv rozdíl mezi stavovou a bezstavovou filtrací není příliš patrný, opět došlo k mírnému zlepšení.

5.8.4. nftables

Bez filtrace

Při testu nftables bez filtrace bylo dosaženo mírně lepšího výsledku než u netfilteru s `iptables`. Bylo provedeno o cca 5 000 transakcí více. Packetová rychlost dosáhla průměru 117 746 packetů/s, přenosová rychlost pak průměru 1,39 Gbit/s a maxima 1,58 Gbit/s (viz také graf A.84).

Stejně jako v předchozích případech, došlo k vyfiltrování správného provozu, ovšem i k nechtěným zahozením (více v grafu A.83).

Filtrace IP adres

I v případě filtrace IP adres došlo k mírnému zlepšení oproti předchozímu nástroji. Přenosová rychlost se průměrně pohybovala okolo 1,01 Gbit/s (maximálně dosáhla 1,21 Gbit/s). Generování packetů se provádělo průměrně rychlostí 102 361 packetů/s. Provedeno bylo celkem 275 152 transakcí.

Úspěšnost filtrace ukazuje graf A.85. Časové průběhy rychlostí jsou k dispozici v grafu A.86.

Filtrace UDP portů

Při filtraci UDP portů bylo tentokrát dosaženo mírně horšího výsledku než u netfilteru a `iptables`. Bylo vyzkoušeno méně transakcí (přesně 370 753 transakcí, což je o cca 6 000 transakcí méně). Průměrná přenosová rychlost byla 1,35 Gbit/s a packetová rychlost se průměrně pohybovala okolo 114 582 packetů/s. Časové průběhy ukazuje graf A.88.

Opět došlo k mírnému zahození provozu, který měl projít bez větších problémů (viz graf A.87).

Filtrace TCP/UDP

V případě filtrace TCP a UDP provozu a zaplnění filtrovací tabulky jedním stem bezvýznamných pravidel, nebylo dosaženo lepších nebo stejných hodnot jako u předchozího filtračního nástroje. Při celkově vyzkoušených 235 699 transakcích se držela přenosová rychlost průměrně okolo 803,32 Mbit/s (maximum bylo 844,44 Mbit/s) a packetová rychlost okolo 68 043 packetů/s.

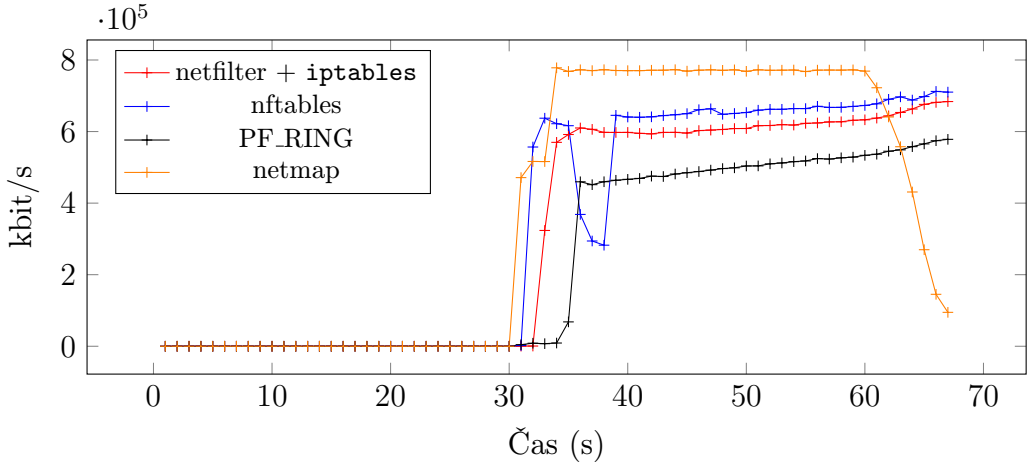
Výsledek úspěšnosti filtrace shrnuje graf A.89. K dipozici je také graf A.90 zobrazující přenosovou a packetovou rychlost v čase.

5.9. Vyhodnocení výsledků z experimentálního pracoviště

Z uvedeného je zřejmé, že většina nástrojů filtruje správně podle nastavených pravidel (jen při jednom testu PF_RINGu prošlo pár nechtěných packetů, což je zvláštní, ale tak to vyhodnotil Spirent Avalanche 3100B). K zahazování ostatního provozu docházelo u všech testovaných firewallů – u některých méně, u některých více. Pro lepší porovnání zkoušených nástrojů, bude rozumné si data z jednotlivých grafů zobrazit do jednoho srovnávacího grafu.

Chceme-li porovnat výkonnost filtrů vůči sobě, dostaneme jedny z nejzajímavějších informací z testu, kdy bylo použito 100 bezvýznamných pravidel + filtrace TCP a UDP komunikací pro vybrané dvojice IP adres. Graf 5.48 ukazuje srovnání přenosových rychlostí při zmíněném (bezstavovém) testu při vyšší rychlosti. Je vidět, že netmap potvrdil svůj dobrý výkon z předešlých testů, a při této zkoušce dosáhl nejlepšího výsledku. Ačkoliv u nftables došlo v jednu chvíli k razantnímu poklesu přenosové rychlosti, přesto se jim podařilo dosáhnout druhé nejvyšší přenosové rychlosti. Netfilter s `iptables` si také nevedl špatně, rozdíl proti nftables není tak velký. Nejhůře se v tomto testu projevil PF_RING, který nedokázal netmapu konkurovat. Zajímavé je, že pořadí nástrojů odpovídá pořadí začátků přenášení dat jednotlivými firewally. Netmap začal přenášet data ihned po 30. sekundě, kdežto PF_RING až o několik sekund později. Netmap také jako jediný kopíroval průběh generování dat, který byl nastaven na Avalanchi. Ostatní nástroje pokračovaly v přenášení dat i po 60. sekundě. Nejspíš je to z toho důvodu, že simulovaní uživatelé zanikaly pomaleji (protože se data přenášely pomaleji), ovšem vytváření simulovaných uživatelů probíhalo podle předem daného průběhu. To by vysvětlovalo i pozvolný nárůst přenosových rychlostí v grafu.

Také tabulka 5.5, srovnávající naměřené hodnoty při uvedeném testu nástrojů, dokazuje výkonnostní pořadí z grafu 5.48. Dobře je to vidět například na procentuálním vy-



Graf 5.48: Srovnání přenosových rychlostí při bezstavovém testu „Filtrace TCP/UDP“ při vyšší přenosové rychlosti.

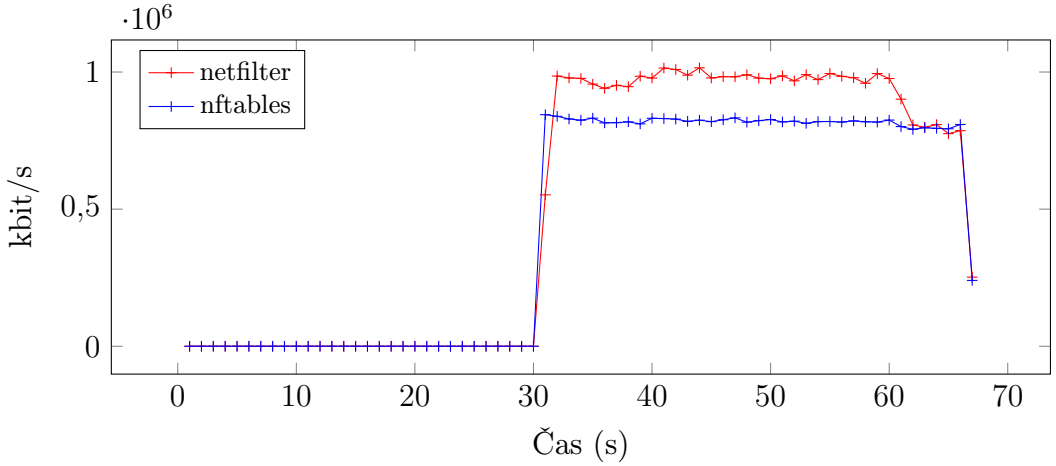
jádření neúspěšných transakcí – zatímco při zkoušení netmapu bylo zahozeno jen 0,49 % transakcí, u nftables to bylo už 3,60 %, netfilter zahodil 8,49 % a při testování PF_RINGu došlo dokonce k 12,58% neúspěšnosti.

Název údaje		netfilter + `iptables`	nftables	PF_RING	netmap
Celkem vyzkoušených transakcí		2 197 419	2 350 233	1 719 504	2 579 755
Úspěšných transakcí		2 010 784	2 265 541	1 503 233	2 567 057
Neúspěšných transakcí		186 635	84 692	216 271	12 698
Podíl úspěšných transakcí (%)		91,51	96,40	87,42	99,51
Podíl neúspěšných transakcí (%)		8,49	3,60	12,58	0,49
Přenosová rychlost (Mbit/s)	minimální	0,338	2,01	4,49	471,29
	maximální	632,87	672,92	533,57	778,15
	průměrná	558,03	594,22	415,33	744,30
Packetová rychlost (packet/s)	minimální	661	1 539	2 270	55 353
	maximální	73 077	74 953	60 184	88 166
	průměrná	65 168	66 865	47 860	84 418

Tabulka 5.5: Srovnávací tabulka statistických údajů pro graf 5.48. Jedná se o měření s filtrací, proto je určitý počet neúspěšných transakcí v pořádku. Informace jsou počítány z hodnot od 30. do 60. sekundy.

Agregace síťových rozhraní není ve speciálních filtračních nástrojích příliš podporována. Firewally, které využívají síťového zásobníku linuxového jádra, nemusejí tuto funkcionalitu zvlášť implementovat, protože pro Linux již řešení umožňující seskupovat síťová rozhraní existují. V opačném případě je nutné, aby byla tato funkce speciálně implementována, což se ale spíše neděje.

Stejně jako v předchozím případě, i pro srovnání testů při linkové agregaci je nejzajímavější porovnání grafů při bezstavovém měření „Filtrace TCP/UDP“ při vyšší přenosové rychlosti. Z grafu 5.49 vyplývá, že při agregaci nejsou nftables tak výkonné jako netfilter s `iptables`. Netfilter dosáhl průměrně o cca 200 Mbit/s více než nftables.



Graf 5.49: Srovnání přenosových rychlostí při bezstavovém testu „Filtrace TCP/UDP“ při vyšší přenosové rychlosti s agregovanými linkami.

V tabulce 5.6 jsou shrnuty různé statistické hodnoty, získané při výše uvedeném měření s agregací linek. Jak je vidět na většině hodnot, vedl si netfilter v tomto případě opravdu lépe – pro příklad lze uvést, že při zkoušení netfilteru bylo vyzkoušeno o cca 65 000 transakcí více než v případě nftables.

Název údaje		netfilter + `iptables`	nftables
Celkem vyzkoušených transakcí		300 377	235 699
Úspěšných transakcí		299 477	234 816
Neúspěšných transakcí		900	883
Podíl úspěšných transakcí (%)		99,70	99,63
Podíl neúspěšných transakcí (%)		0,30	0,37
Přenosová rychlost (Mbit/s)	minimální	98,77	240,10
	maximální	1 190,00	844,44
	průměrná	1 030,00	803,32
Packetová rychlost (packet/s)	minimální	8 250	20 347
	maximální	100 613	70 991
	průměrná	86 904	68 043

Tabulka 5.6: Srovnávací tabulka statistických údajů pro graf 5.49.

5.10. Celkové zhodnocení

Jak je možné ze všech předchozích odstavců a výsledků vytušit, univerzální a dokonalý nástroj pro vysokorychlostní filtrování síťového provozu na běžně dostupném hardwaru a softwaru zřejmě neexistuje. Každý ze zkoušených nástrojů má svá pro a proti, a pro každý jednotlivý případ použití je v současné době potřeba vyhodnotit, který nástroj bude vhodnější použít. Netfilter s `iptables` například implementují velké množství funkcí včetně stavové filtrace a agregace síťových rozhraní. Ovšem jak bylo ukázáno v předchozím textu, při vyšších přenosových rychlostech se již projevuje menší výkonnost oproti speciálním nástrojům jako je netmap (s upravenými ovladači NIC). Ten ovšem zase nepodporuje funkce, které by mohly být v určitých situacích přínosné, a je možné jej využít více-

méně pouze k bezstavové filtraci a tvarování provozu. Určitým kompromisem mohou být nftables, které sice také (zřejmě zatím) nepodporují hlídání stavu spojení, ovšem díky podpoře linuxového jádra, umožňují alespoň agregaci síťových rozhraní. Také bylo ukázáno, že jsou nftables výkonnější než netfilter s `iptables` a do budoucna rozhodně perspektivnější. Ukázková aplikace PF_RINGu, umožňující přemostění dvou rozhraní a filtraci, je opravdu spíše na ukázku možností frameworku PF_RING než na seriózní použití, a proto se nedá očekávat bohatá funkcionalita. Tabulka 5.7 obsahuje shrnutí vlastností a funkcí, které byly o filtrovacích nástrojích zjištěny v průběhu zpracovávání této práce.

Funkce, vlastnost	netfilter + `iptables`	nftables	PF_RING	netmap
Bezstavová filtrace	✓	✓	✓	✓
Stavová filtrace	✓	✗	✗	✗[1]
Podpora 10 Gbit/s	✗	✗	✗[4]	✓
Změna pravidel za běhu	✓	✓	✗	✓
Agregace síťových rozhraní	✓	✓	✗	✗
Podpora IPv6[3]	✓	✓	✓	✗[2]
Tvarování provozu[3]	✓	✓	✗	✓

Tabulka 5.7: Tabulka vlastností a funkcí zkoušených nástrojů.

[1] Nástroj umožňuje použít příkaz s touto funkcí, ovšem funkce se na provoz neaplikuje.
[2] Nápověda nástroje tuto funkci obsahuje, ovšem po zadání příkazu se vypíše chyba.
[3] Netestováno na procházejícím provozu.
[4] Testovaná nelicencovaná verze.

6. Závěr

Cílem této práce bylo nalézt nástroje a knihovny, pomocí nichž by bylo možné filtrovat síťový provoz na platformě Linux a to na běžně dostupném hardwaru. Práce se zaměřovala na vysokorychlostní transparentní filtraci, aby bylo možné propouštět určitý provoz až na 10Gbit/s síti bez zásahu do adresování dané sítě. Po úvodu byly rozebrány různé pohledy na síťové filtry (firewally) – jejich rozdělení a obvyklé umístění v síti. Práce se přitom soustředila hlavně na transparentní síťovou packetovou filtraci. Dále byla v krátkosti představena běžná metoda zpracování příchozího síťového provozu na Linuxu a uvedeny možnosti, jakým způsobem lze dosáhnout vyšší výkonnosti v této oblasti. Tato část také představila různé nástroje a knihovny, které lze na Linuxu k filtraci (vysokorychlostního) síťového provozu využít. Následovalo samotné praktické testování vybraných nástrojů. Po popisu testovacího prostředí a testovacích metod, byly vybrané nástroje v testovacím prostředí nainstalovány a odzkoušeny jejich filtrovací možnosti a výkonnost. Testování proběhlo v několika fázích – nejdříve při rychlosti do 1 Gbit/s, později přímo pro rychlost téměř 10 Gbit/s. Následně byly nástroje otestovány v experimentální síti, kde byla vyzkoušena i agregace síťových rozhraní. Z výsledků je zřejmé, že zatímco s rychlostmi stovek Mbit/s si bez problémů poradí většina nástrojů (i standardní linuxové vybavení), pro filtraci na rychlosti kolem 10 Gbit/s je již potřeba software speciálně upravený pro tyto účely, který ovšem nemusí být tak funkčně vybavený jako standardní nástroje.

Osobně vidím v této oblasti velký potenciál v oblasti vývoje a inovací. Například PF_RING sice nabízí aplikaci pro transparentní filtraci, ovšem jedná se spíše o ukázku nebo prototyp než o plnohodnotný firewall. Aby bylo možné přidat pravidlo, je potřeba přemostění zastavit a pustit s upraveným parametrem. Firewally běžně používají tabulku pravidel, kde je možné měnit pravidla za běhu. `netmap-ipfw` již tento přístup nabízí, ovšem k dokonalosti má taktéž daleko. Ačkoliv utilita pro správu pravidel nabízí možnost pro přidávání pravidel pro hlídání stavu spojení, ve skutečnosti tato funkce nefunguje a při průchodu provozu se pravidla neaplikují. Prostor pro vylepšení je také v oblasti výkonu. Momentálně je filtrovací démon `kipfw` pouze jednovláknový a jednoprocesový. V dnešní době, kdy jsou k dispozici vícejádrové procesory, by jistě šlo například zpracování provozu na více rozhraních zparalelizovat a dosáhnout tak vyšší datové propustnosti.

Literatura

[1] BURDA, K.: *Návrh, správa a bezpečnost počítačových sítí*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 171 s.

[2] CISCO SYSTEMS, INC.: *PIX/ASA: Transparent Firewall Configuration Example* [online]. [cit. 2016-10-25]. Dostupné z: `http://www.cisco.com/c/en/us/support/docs/security/pix-500-series-security-appliances/97853-Transparent-firewall.html`

[3] GALLENMÜLLER, S.: *Comparison of Memory Mapping Techniques for High-Speed Packet Processing*. [Diplomová práce.] München: Technische Universität München, 2014. 63 s.

[4] SALIM, J. H.: *When NAPI Comes To Town* [online]. 2005 [cit. 2016-10-29]. Dostupné z: `ftp://ftp.dei.uc.pt/pub/linux/kernel/people/hadi/docs/UKUUG2005.pdf`

[5] GALLENMÜLLER, S., et al.: *Comparison of Frameworks for High-Performance Packet IO*. München: Technische Universität München, Department of Informatics, 2015. 10 s.

[6] PEŠA, D.: *Pokročilé metody filtrování síťového provozu v Linuxu*. [Diplomová práce]. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 71 s.

[7] NETFILTER.ORG: *The netfilter.org „nftables“ project* [online]. [cit. 2016-10-29]. Dostupné z: `https://www.netfilter.org/projects/nftables/index.html`

[8] NTOP.ORG: *PF_RING User Guide* [online]. 2016 [cit. 2017-05-21]. 26 s. Dostupné z: `https://github.com/ntop/PF_RING/blob/dev/doc/PF_RING-UsersGuide.pdf`

[9] NTOP.ORG: *PF_RING ZC (Zero Copy)* [online]. [cit. 2016-10-29]. Dostupné z: `http://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/`

[10] DPDK: *Supported NICs* [online]. [cit. 2016-10-30]. Dostupné z: `http://dpdk.org/doc/nics`

[11] BROCADE, INC.: *Brocade vRouter Data Sheet* [online]. 2017 [cit. 2017-05-22]. 10 s. Dostupné z: `https://www.brocade.com/content/dam/common/documents/content-types/datasheet/brocade-vrouter-ds.pdf`

[12] 6WIND: *6WIND Turbo Router Data Sheet* [online]. [cit. 2016-10-30]. 2 s. Dostupné z: `http://www.6wind.com/wp-content/uploads/2016/10/Turbo-Router-Data-Sheet.pdf`

[13] SNABBWALL: *Roadmap* [online]. [cit. 2016-10-30]. Dostupné z: `http://snabbwall.org/roadmap/`

[14] HIPAC: *FAQ* [online]. [cit. 2016-10-30]. Dostupné z: `http://www.hipac.org/documentation/faq.html`

[15] HIPAC: *Installation requirements* [online]. [cit. 2016-10-30]. Dostupné z: `http://www.hipac.org/documentation/installation.html`

[16] ŠOUN, J.: *Redundance v datových sítích.* [Diplomová práce]. Praha: Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, 2012. 63 s.

[17] RED HAT, INC.: *5.3. Comparison of Network Teaming to Bonding* [online]. [cit. 2016-12-11]. Dostupné z: `https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Networking_Guide/sec-Comparison_of_Network_Teaming_to_Bonding.html`

[18] DPDK: *10. Link Bonding Poll Mode Driver Library* [online]. [cit. 2016-12-11]. Dostupné z: `http://dpdk.org/doc/guides-16.07/prog_guide/link_bonding_poll_mode_drv_lib.html`

[19] SPIRENT COMMUNICATIONS, INC.: *Spirent Avalanche 3100B: Application load testing solution* [online]. 2011. 4 s. Dostupné z: `https://www.infopoint-security.de/medien/spirent_avalanche_3100_datasheet.pdf`

[20] TCPDUMP: *Berkeley Packet Filter (BPF) syntax* [online]. [cit. 2016-12-05]. Dostupné z: `https://biot.com/capstats/bpf.html`

[21] NFTABLES WIKI: *Bridge filtering* [online]. [cit. 2017-05-10]. Dostupné z: `https://wiki.nftables.org/wiki-nftables/index.php/Bridge_filtering`

[22] DAVIS T., et al.: *Linux Ethernet Bonding Driver HOWTO* [online]. [cit. 2017-05-12]. Dostupné z: `https://www.kernel.org/doc/Documentation/networking/bonding.txt`

7. Seznam použitých zkratek a symbolů

API	Application Programming Interface
ARP	Address Resolution Protocol
ASIC	Application-Specific Integrated Circuit
BPF	Berkeley Packet Filter
BSD	Berkeley Software Distribution
DNS	Domain Name System
FPGA	Field-Programmable Gate Array
GPL	General Public License
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IO	Input/Output
IP	Internet Protocol
IRQ	Interrupt Request
ISO	International Organization for Standardization
JIT	Just-In-Time
LACP	Link Aggregation Control Protocol
LGPL	Lesser General Public License
MAC	Media Access Control
NAPI	New API
NIC	Network Interface Controller
OSI	Open Systems Interconnection
QoS	Quality of Service
RAM	Random Access Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UIO	User Input/Output
URL	Uniform Resource Locator
UTP	Unshielded Twisted Pair
VLAN	Virtual Local Area Network

A. Přílohy

A.1. Tabulky s výsledky první fáze měření

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	828 129	828 129	0
ICMP	276 043	276 043	0
rozsah1			
HTTP	138 022	138 022	0
DNS	138 022	138 022	0
rozsah2			
HTTP	138 021	138 021	0
DNS	138 021	138 021	0

Tabulka A.1: Výsledky referenčního měření testování do rychlosti 1 Gbit/s.

A.1.1. netfilter + iptables

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	857 913	857 234	679
ICMP	285 971	285 292	679
rozsah1			
HTTP	142 986	142 986	0
DNS	142 986	142 986	0
rozsah2			
HTTP	142 985	142 985	0
DNS	142 985	142 985	0

Tabulka A.2: netfilter + `iptables` – bez filtrace.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	908 529	570 621	337 908
ICMP	285 315	285 311	4
rozsah1			
HTTP	168 952	0	168 952
DNS	142 656	142 656	0
rozsah2			
HTTP	168 951	0	168 951
DNS	142 655	142 654	1

Tabulka A.3: netfilter + `iptables` – filtrace HTTP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	861 257	556 433	304 824
ICMP	284 732	273 388	11 344
rozsah1			
HTTP	145 898	141 509	4 389
DNS	142 353	0	142 353
rozsah2			
HTTP	145 898	141 536	4 362
DNS	142 376	0	142 376

Tabulka A.4: netfilter + `iptables` – filtrace DNS.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	876 399	583 578	292 821
ICMP	292 233	0	292 233
rozsah1			
HTTP	146 288	146 106	182
DNS	145 793	145 681	112
rozsah2			
HTTP	146 288	146 116	172
DNS	145 797	145 675	122

Tabulka A.5: netfilter + `iptables` – filtrace ICMP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	849 286	499 056	350 230
ICMP	261 961	166 348	95 613
rozsah1			
HTTP	166 354	166 354	0
DNS	166 354	166 354	0
rozsah2			
HTTP	166 353	0	166 353
DNS	88 264	0	88 264

Tabulka A.6: netfilter + `iptables` – filtrace rozsahu IP adres.

A.1.2. nftables

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	859 374	859 336	38
ICMP	286 458	286 420	38
rozsah1			
HTTP	143 229	143 229	0
DNS	143 229	143 229	0
rozsah2			
HTTP	143 229	143 229	0
DNS	143 229	143 229	0

Tabulka A.7: nftables – bez filtrace.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	910 823	570 995	339 828
ICMP	285 504	285 499	5
rozsah1			
HTTP	169 910	0	169 910
DNS	142 750	142 748	2
rozsah2			
HTTP	169 910	0	169 910
DNS	142 749	142 748	1

Tabulka A.8: nftables – filtrace HTTP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	858 445	572 086	286 359
ICMP	286 134	285 952	182
rozsah1			
HTTP	143 089	143 068	21
DNS	143 068	0	143 068
rozsah2			
HTTP	143 088	143 066	22
DNS	143 066	0	143 066

Tabulka A.9: nftables – filtrace DNS.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	882 908	588 592	294 316
ICMP	294 298	0	294 298
rozsah1			
HTTP	147 158	147 148	10
DNS	147 146	147 146	0
rozsah2			
HTTP	147 158	147 150	8
DNS	147 148	147 148	0

Tabulka A.10: nftables – filtrace ICMP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	849 871	499 392	350 479
ICMP	262 141	166 464	95 677
rozsah1			
HTTP	166 464	166 464	0
DNS	166 464	166 464	0
rozsah2			
HTTP	166 464	0	166 464
DNS	88 338	0	88 338

Tabulka A.11: nftables – filtrace rozsahu IP adres.

A.1.3. PF_RING

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	858 942	858 928	14
ICMP	286 314	286 300	14
rozsah1			
HTTP	143 157	143 157	0
DNS	143 157	143 157	0
rozsah2			
HTTP	143 157	143 157	0
DNS	143 157	143 157	0

Tabulka A.12: PF_RING – bez filtrace.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	909 901	570 650	339 251
ICMP	285 333	285 326	7
rozsah1			
HTTP	169 621	0	169 621
DNS	142 663	142 661	2
rozsah2			
HTTP	169 621	0	169 621
DNS	142 663	142 663	0

Tabulka A.13: PF_RING – filtrace HTTP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	858 177	571 991	286 186
ICMP	286 059	285 932	127
rozsah1			
HTTP	143 030	143 030	0
DNS	143 030	0	143 030
rozsah2			
HTTP	143 029	143 029	0
DNS	143 029	0	143 029

Tabulka A.14: PF_RING – filtrace DNS.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	882 977	588 649	294 328
ICMP	294 328	0	294 328
rozsah1			
HTTP	147 164	147164	0
DNS	147 164	147 164	0
rozsah2			
HTTP	147 164	147 164	0
DNS	147 160	147 160	0

Tabulka A.15: PF_RING – filtrace ICMP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	849 863	499 385	350 478
ICMP	262 143	166 461	95 683
rozsah1			
HTTP	166 462	166 462	0
DNS	166 462	166 462	0
rozsah2			
HTTP	166 461	0	166 461
DNS	88 335	0	88 335

Tabulka A.16: PF_RING – filtrace rozsahu IP adres.

A.1.4. netmap

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	856 302	856 163	139
ICMP	285 434	285 295	139
rozsah1			
HTTP	142 717	142 717	0
DNS	142 717	142 717	0
rozsah2			
HTTP	142 717	142 717	0
DNS	142 717	142 717	0

Tabulka A.17: netmap – bez filtrace.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	913 731	570 574	343 157
ICMP	285 292	285 288	4
rozsah1			
HTTP	171 576	0	171 576
DNS	142 644	142 643	1
rozsah2			
HTTP	171 575	0	171 575
DNS	142 644	142 643	1

Tabulka A.18: netmap – filtrace HTTP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	854 382	567 572	284 810
ICMP	284 794	284 778	16
rozsah1			
HTTP	142 397	142 397	0
DNS	142 397	0	142 397
rozsah2			
HTTP	142 397	142 397	0
DNS	142 397	0	142 397

Tabulka A.19: netmap – filtrace DNS.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	885 527	590 342	295 185
ICMP	295 184	0	295 184
rozsah1			
HTTP	147 592	147 592	0
DNS	147 580	147 579	1
rozsah2			
HTTP	147 592	147 592	0
DNS	147 579	147 579	0

Tabulka A.20: netmap – filtrace ICMP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	849 064	499 056	350 008
ICMP	261 851	166 352	95 499
rozsah1			
HTTP	166 352	166 352	0
DNS	166 352	166 352	0
rozsah2			
HTTP	166 351	0	166 351
DNS	88 158	0	88 158

Tabulka A.21: netmap – filtrace rozsahu IP adres.

A.2. Tabulky s výsledky druhé fáze měření

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	1 028 022	1 028 022	0
ICMP	102 940	102 940	0
rozsah1			
HTTP	411 702	411 702	0
DNS	51 138	51 138	0
rozsah2			
HTTP	411 354	411 354	0
DNS	50 888	50 888	0

Tabulka A.22: Výsledky referenčního měření testování do rychlosti 10 Gbit/s.

A.2.1. netfilter + iptables

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	740 485	702 174	38 311
ICMP	175 660	174 641	1 019
rozsah1			
HTTP	183 338	164 745	18 593
DNS	99 283	99 196	87
rozsah2			
HTTP	183 392	164 876	18 516
DNS	98 812	98 716	96

Tabulka A.23: netfilter + `iptables` – bez filtrace.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	4 186 139	3 947 103	239 036
ICMP	1 968 857	1 968 587	270
rozsah1			
HTTP	119 245	0	119 245
DNS	989 523	989 387	136
rozsah2			
HTTP	119 243	0	119 243
DNS	989 271	989 129	142

Tabulka A.24: netfilter + `iptables` – filtrace HTTP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	873 565	845 462	28 103
ICMP	111 214	111 201	13
rozsah1			
HTTP	367 230	367 092	138
DNS	13 908	0	13 908
rozsah2			
HTTP	367 314	367 169	145
DNS	13 899	0	13 899

Tabulka A.25: netfilter + `iptables` – filtrace DNS.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	853 717	750 363	103 354
ICMP	75 306	0	75 306
rozsah1			
HTTP	197 528	183 469	14 059
DNS	192 027	192 007	20
rozsah2			
HTTP	196 708	182 759	13 949
DNS	192 148	192 128	20

Tabulka A.26: netfilter + `iptables` – filtrace ICMP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	668 011	579 043	88 968
ICMP	98 018	81 832	16 186
rozsah1			
HTTP	404 048	394 602	9 446
DNS	102 646	102 609	37
rozsah2			
HTTP	48 616	0	48 616
DNS	14 683	0	14 683

Tabulka A.27: netfilter + `iptables` – filtrace rozsahu IP adres.

A.2.2. nftables

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	947 046	945 695	1 351
ICMP	135 376	135 355	21
rozsah1			
HTTP	326 281	325 618	663
DNS	80 450	80 450	0
rozsah2			
HTTP	324 238	323 571	667
DNS	80 701	80 701	0

Tabulka A.28: nftables – bez filtrace.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	4 196 707	3 957 331	239 376
ICMP	1 972 917	1 972 641	276
rozsah1			
HTTP	119 415	0	119 415
DNS	993 033	992 900	133
rozsah2			
HTTP	119 415	0	119 415
DNS	991 927	991 790	137

Tabulka A.29: nftables – filtrace HTTP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	872 852	829 638	43 214
ICMP	250 766	250 574	192
rozsah1			
HTTP	294 710	289 743	4 967
DNS	16 552	0	16 552
rozsah2			
HTTP	294 287	289 321	4 966
DNS	16 537	0	16 537

Tabulka A.30: nftables – filtrace DNS.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	946 319	910 374	35 945
ICMP	28 120	0	28 120
rozsah1			
HTTP	278 387	374 458	3 929
DNS	180 339	180 318	21
rozsah2			
HTTP	278 522	274 679	3 843
DNS	180 951	180 919	32

Tabulka A.31: nftables – filtrace ICMP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	891 088	887 684	3 404
ICMP	56 532	55 892	640
rozsah1			
HTTP	771 073	771 073	0
DNS	60 719	60 719	0
rozsah2			
HTTP	2 175	0	2 175
DNS	589	0	589

Tabulka A.32: nftables – filtrace rozsahu IP adres.

A.2.3. PF_RING

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	610 939	601 792	9 147
ICMP	105 667	104 551	1 116
rozsah1			
HTTP	196 726	192 755	3 971
DNS	53 928	53 885	43
rozsah2			
HTTP	198 913	194 946	3 967
DNS	55 705	55 655	50

Tabulka A.33: PF_RING – bez filtrace.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	3 975 968	3 736 581	239 387
ICMP	1 868 540	1 868 263	277
rozsah1			
HTTP	119 423	0	119 423
DNS	933 923	933 790	133
rozsah2			
HTTP	119 420	0	119 420
DNS	934 662	934 528	134

Tabulka A.34: PF_RING – filtrace HTTP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	591 765	508 554	83 211
ICMP	142 497	137 252	5 245
rozsah1			
HTTP	197 719	185 503	12 216
DNS	26 780	0	26 780
rozsah2			
HTTP	197 804	185 799	12 005
DNS	26 965	0	26 965

Tabulka A.35: PF_RING – filtrace DNS.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	582 493	545 130	37 363
ICMP	26 416	0	26 416
rozsah1			
HTTP	191 015	185 608	5 407
DNS	89 023	88 983	40
rozsah2			
HTTP	190 997	185 542	5 455
DNS	85 042	84 997	45

Tabulka A.36: PF_RING – filtrace ICMP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	535 869	511 605	24 264
ICMP	61 143	55 141	6 002
rozsah1			
HTTP	401 056	399 636	1 420
DNS	56 838	56 828	10
rozsah2			
HTTP	11 402	0	11 402
DNS	5 430	0	5 430

Tabulka A.37: PF_RING – filtrace rozsahu IP adres.

A.2.4. netmap

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	538 346	490 774	47 572
ICMP	81 237	78 217	3 020
rozsah1			
HTTP	186 885	165 637	21 248
DNS	39 902	38 818	1 084
rozsah2			
HTTP	189 906	168 727	21 179
DNS	40 416	39 375	1 041

Tabulka A.38: netmap – bez filtrace.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	2 469 393	2 229 981	239 412
ICMP	1 109 757	1 109 484	273
rozsah1			
HTTP	119 431	0	119 431
DNS	561 522	561 381	141
rozsah2			
HTTP	119 428	0	119 428
DNS	559 225	559 116	139

Tabulka A.39: netmap – filtrace HTTP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	580 873	318 840	262 033
ICMP	143 853	115 784	28 069
rozsah1			
HTTP	148 360	101 478	46 882
DNS	70 142	0	70 142
rozsah2			
HTTP	148 343	101 578	46 765
DNS	70 175	0	70 175

Tabulka A.40: netmap – filtrace DNS.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	621 040	357 319	263 721
ICMP	187 194	0	187 194
rozsah1			
HTTP	129 438	96 416	33 022
DNS	87 664	82 218	5 446
rozsah2			
HTTP	129 415	96 172	33 243
DNS	87 329	82 513	4 816

Tabulka A.41: netmap – filtrace ICMP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	590 549	504 147	86 402
ICMP	103 699	83 576	20 123
rozsah1			
HTTP	351 854	336 420	15 434
DNS	84 176	84 151	25
rozsah2			
HTTP	31 912	0	31 912
DNS	18 908	0	18 908

Tabulka A.42: netmap – filtrace rozsahu IP adres.

A.2.5. netmap (vlastní ovladač)

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	992 073	992 073	0
ICMP	85 177	85 177	0
rozsah1			
HTTP	409 897	409 897	0
DNS	43 828	43 828	0
rozsah2			
HTTP	409 847	409 847	0
DNS	43 384	43 384	0

Tabulka A.43: netmap (vlastní ovladač) – bez filtrace.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	4 197 292	3 957 901	239 391
ICMP	1 976 133	1 975 861	272
rozsah1			
HTTP	119 426	0	119 426
DNS	991 164	991 029	135
rozsah2			
HTTP	119 422	0	119 422
DNS	991 147	991 011	136

Tabulka A.44: netmap (vlastní ovladač) – filtrace HTTP.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	954 180	948 473	5 707
ICMP	110 729	110 729	0
rozsah1			
HTTP	418 997	418 997	0
DNS	2 855	0	2 855
rozsah2			
HTTP	418 747	418 747	0
DNS	2 852	0	2 852

Tabulka A.45: netmap (vlastní ovladač) – filtrace DNS.

Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	938 708	933 905	4 803
ICMP	4 803	0	4 803
rozsah1			
HTTP	416 678	416 678	0
DNS	50 525	50 525	0
rozsah2			
HTTP	416 479	416 479	0
DNS	50 223	50 223	0

Tabulka A.46: netmap (vlastní ovladač) – filtrace ICMP.

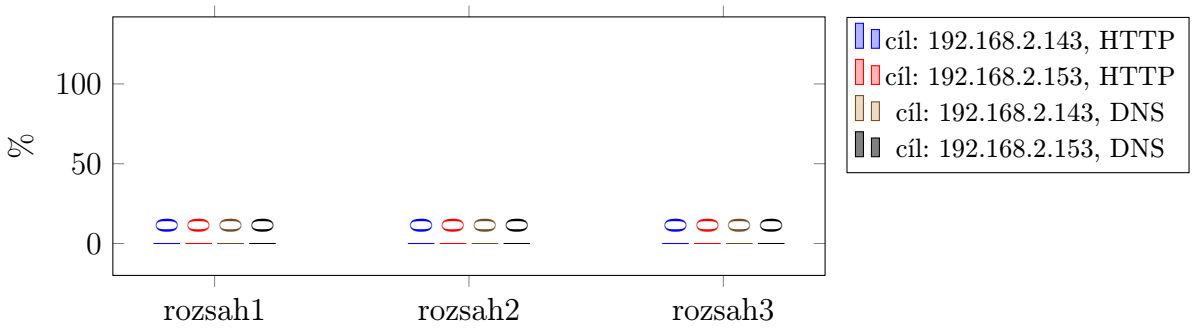
Název	Počet vyzkoušených transakcí	Počet úspěšných transakcí	Počet neúspěšných transakcí
Celkově	934 869	932 768	2 101
ICMP	56 767	56 245	522
rozsah1			
HTTP	819 281	819 281	0
DNS	57 242	57 242	0
rozsah2			
HTTP	1 095	0	1 095
DNS	484	0	484

Tabulka A.47: netmap (vlastní ovladač) – filtrace rozsahu IP adres.

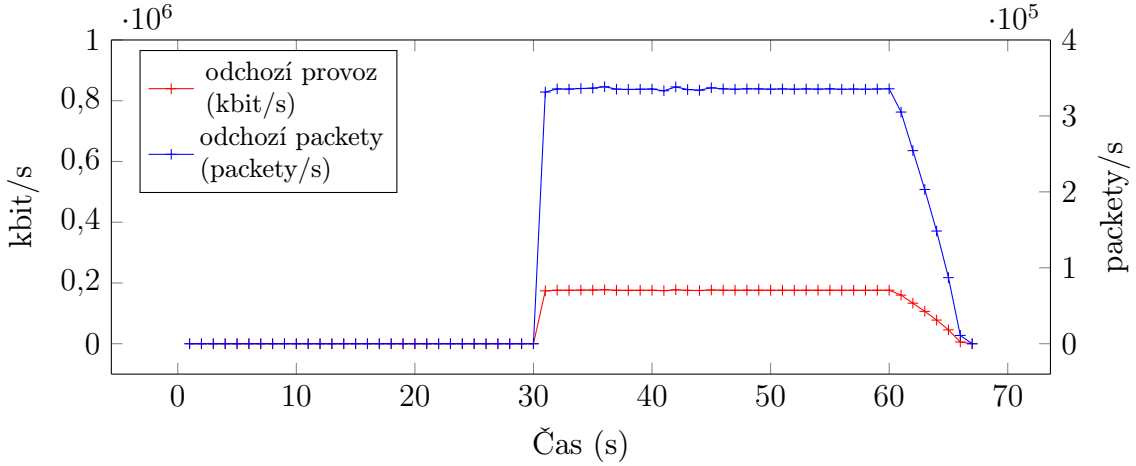
A.3. Grafy z měření v experimentální síti

A.3.1. netfilter + iptables

Bez filtrace, nižší rychlost

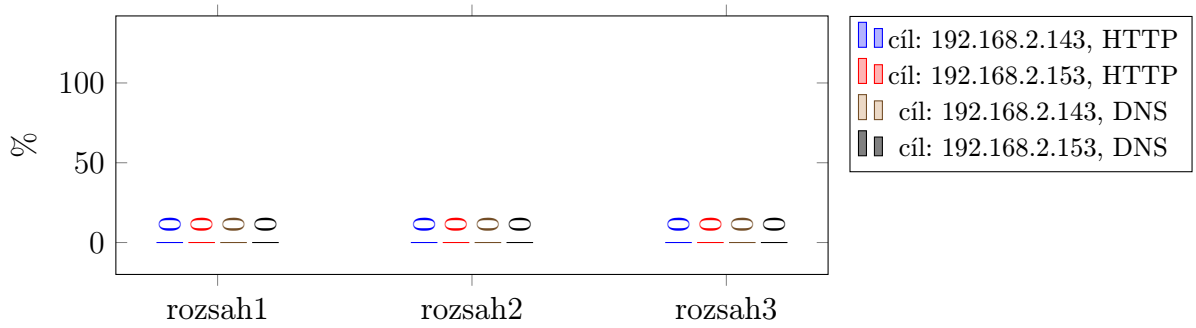


Graf A.1: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – bez filtrace, nižší rychlost.

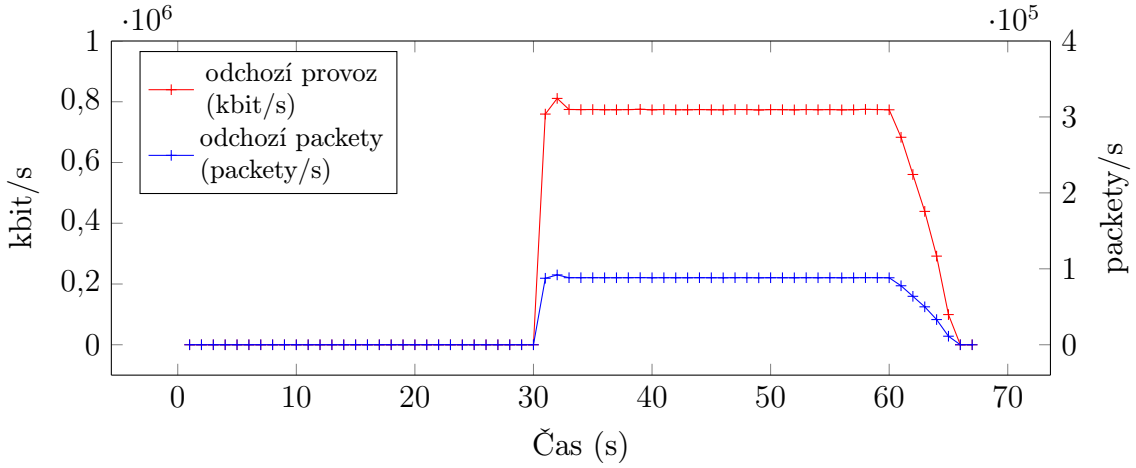


Graf A.2: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – bez filtrace, nižší rychlost.

Bez filtrace, vyšší rychlost

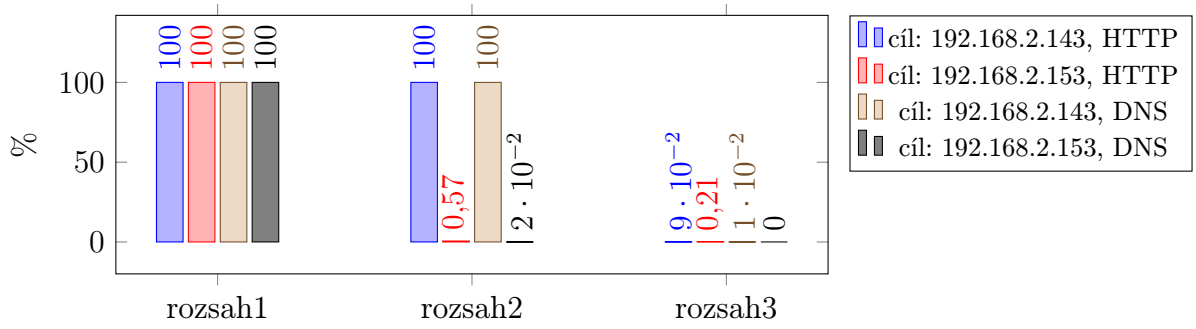


Graf A.3: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – bez filtrace, vyšší rychlost.

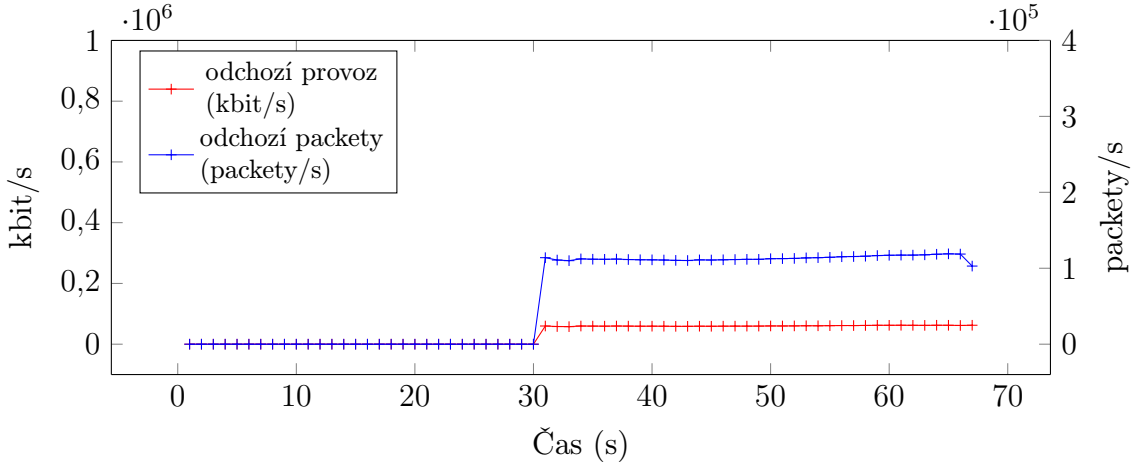


Graf A.4: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – bez filtrace, vyšší rychlost.

Filtrace IP adres, nižší rychlost

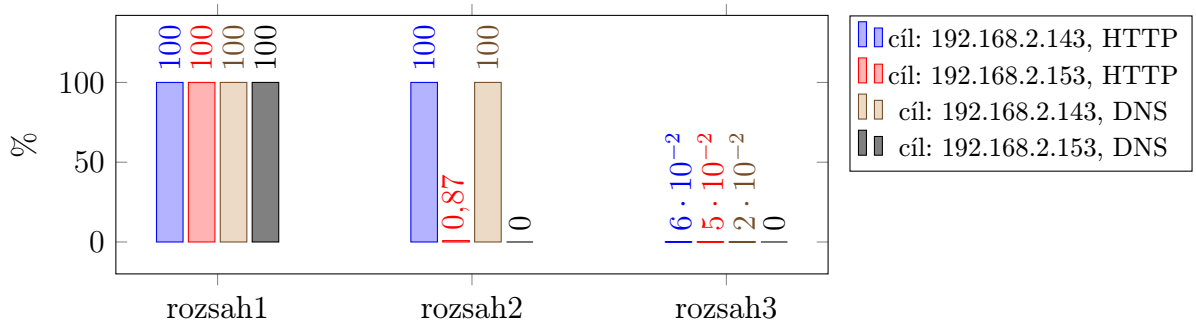


Graf A.5: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace IP adres, nižší rychlost.

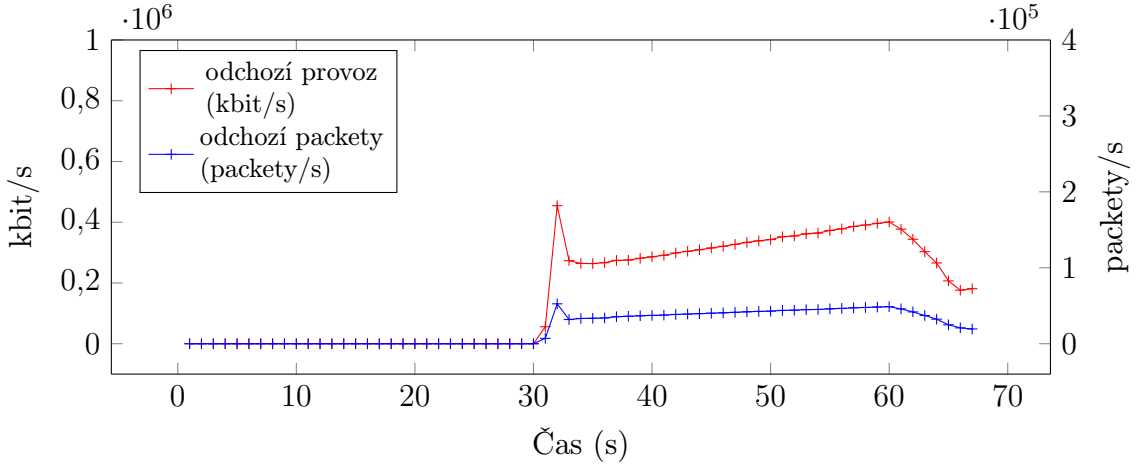


Graf A.6: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace IP adres, nižší rychlost.

Filtrace IP adres, vyšší rychlost

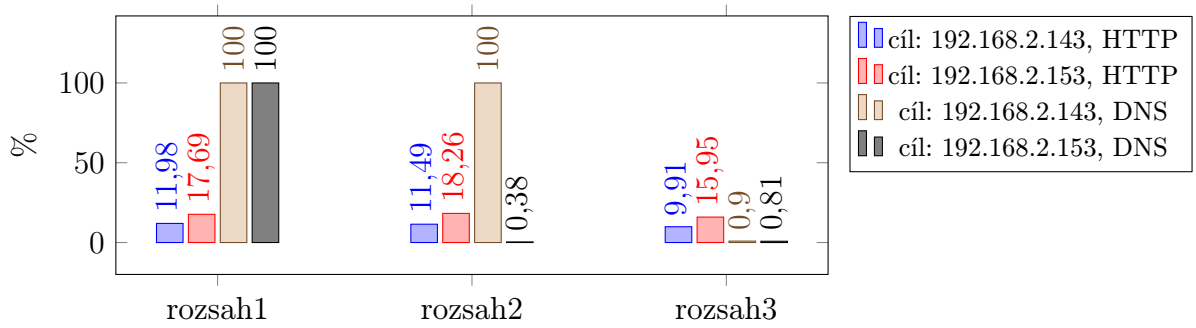


Graf A.7: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace IP adres, vyšší rychlost.

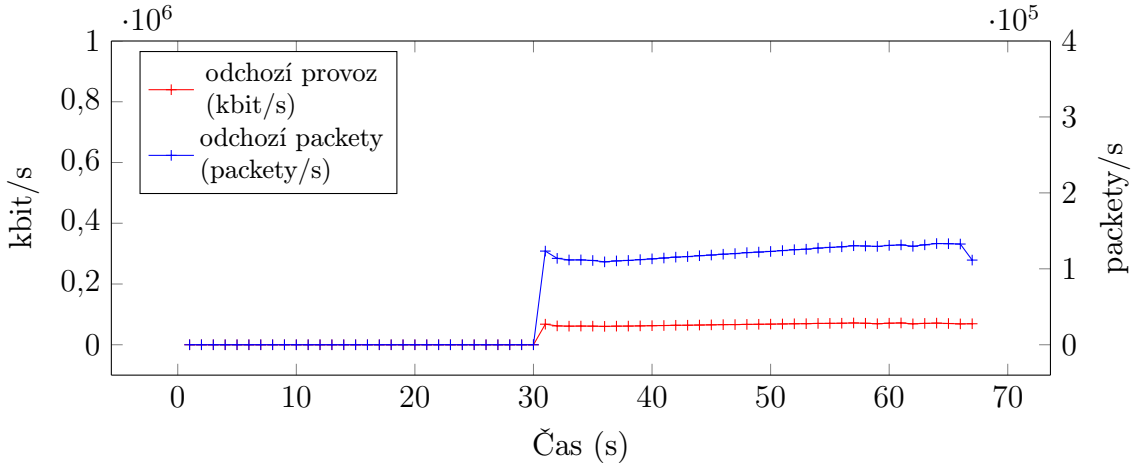


Graf A.8: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace IP adres, vyšší rychlost.

Filtrace UDP portů, nižší rychlost

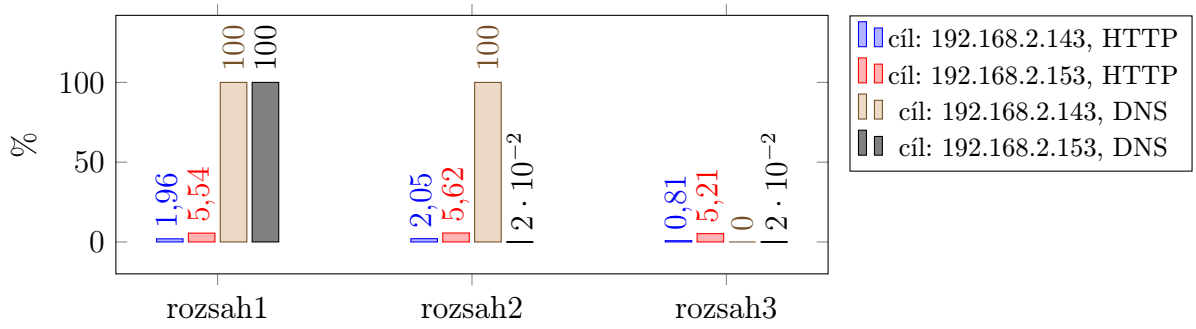


Graf A.9: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace UDP portů, nižší rychlost.

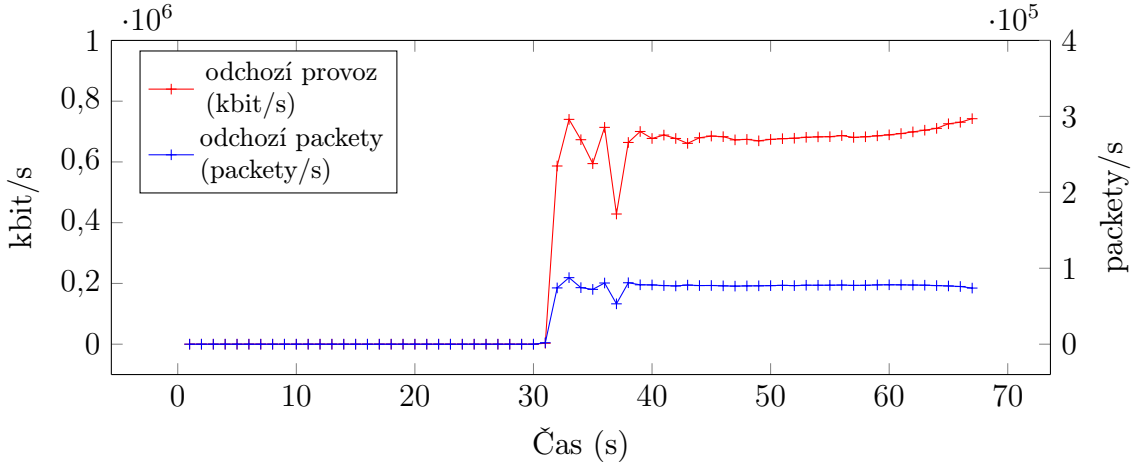


Graf A.10: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace UDP portů, nižší rychlost.

Filtrace UDP portů, vyšší rychlost

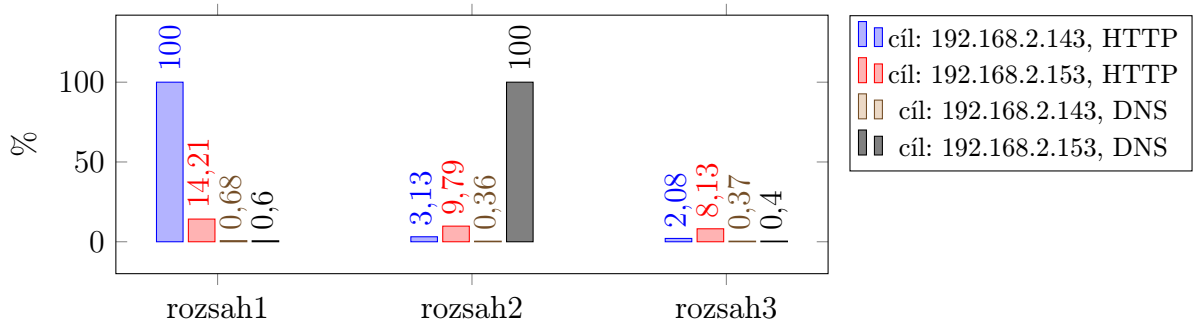


Graf A.11: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace UDP portů, vyšší rychlost.

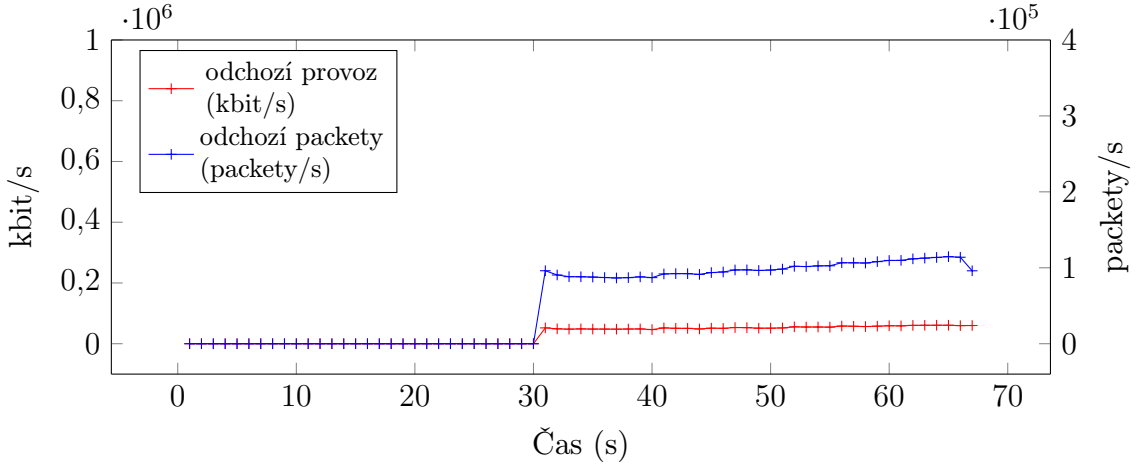


Graf A.12: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace UDP portů, vyšší rychlost.

Filtrace TCP/UDP, nižší rychlost, bezstavová

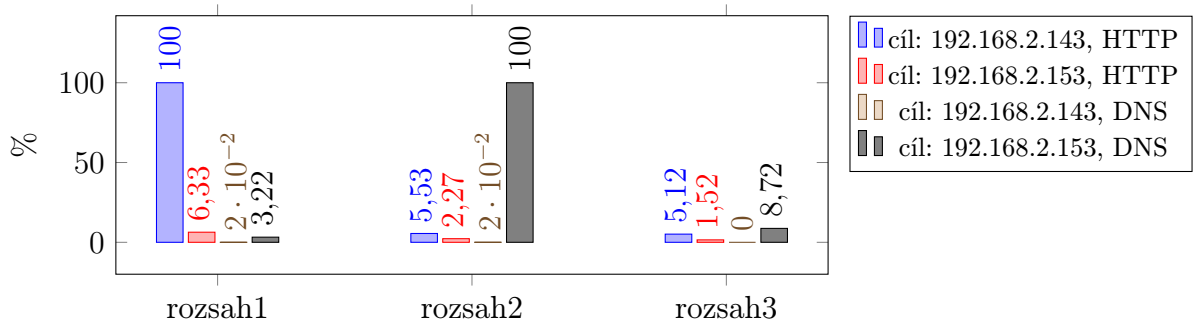


Graf A.13: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace TCP/UDP, nižší rychlost, bezstavová.

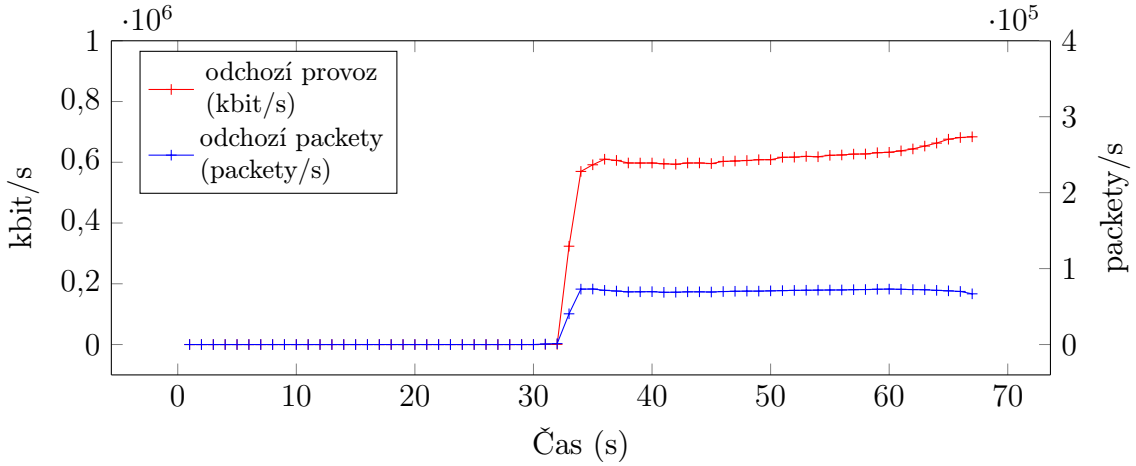


Graf A.14: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace TCP/UDP, nižší rychlost, bezstavová.

Filtrace TCP/UDP, vyšší rychlost, bezstavová

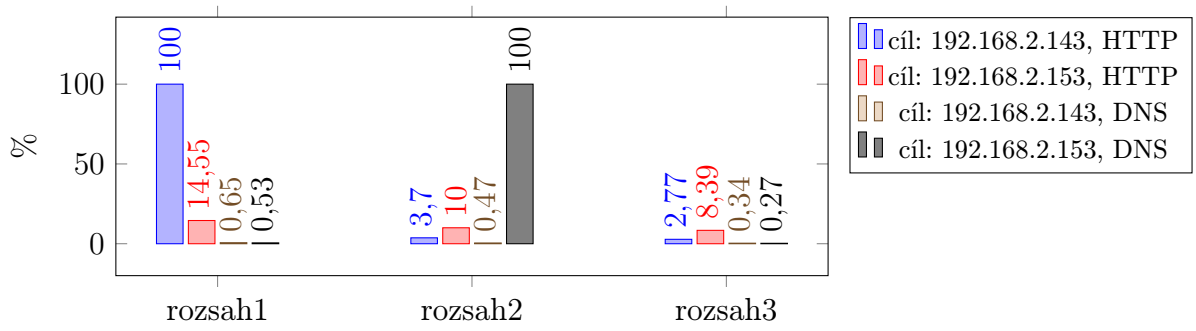


Graf A.15: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace TCP/UDP, vyšší rychlost, bezstavová.

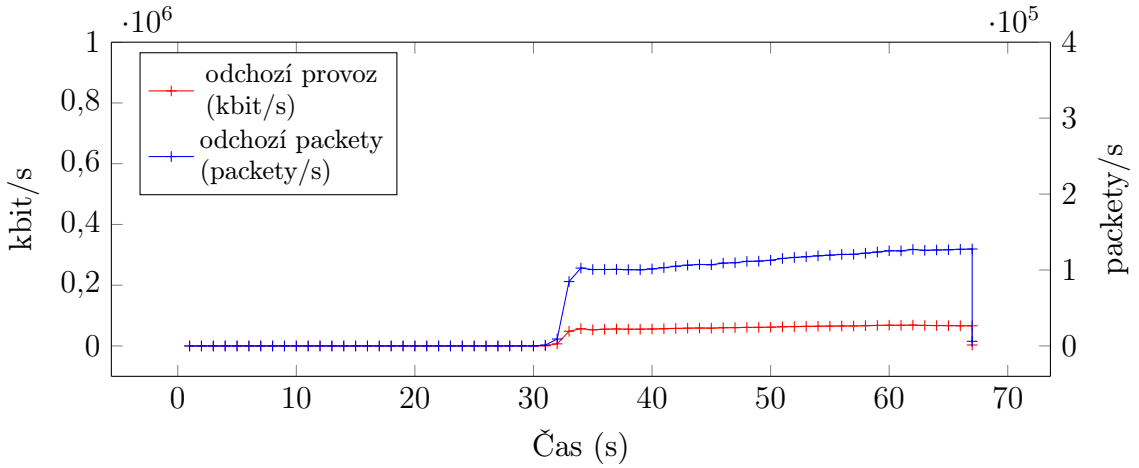


Graf A.16: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace TCP/UDP, vyšší rychlost, bezstavová.

Filtrace TCP/UDP, nižší rychlost, stavová

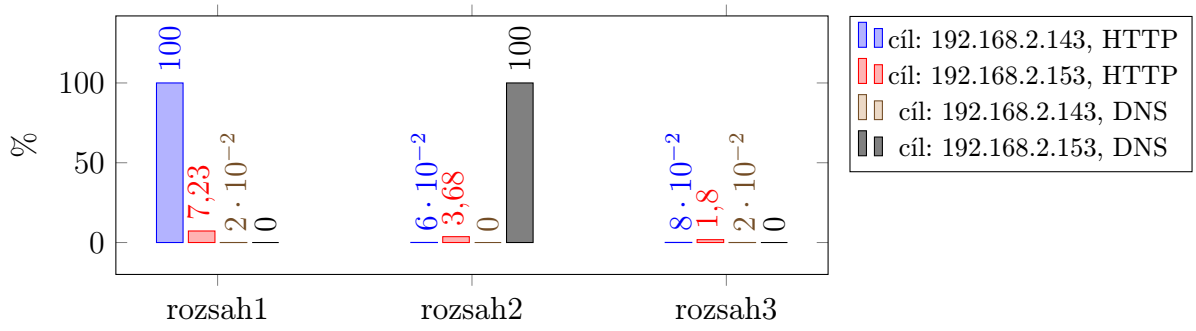


Graf A.17: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace TCP/UDP, nižší rychlost, stavová.

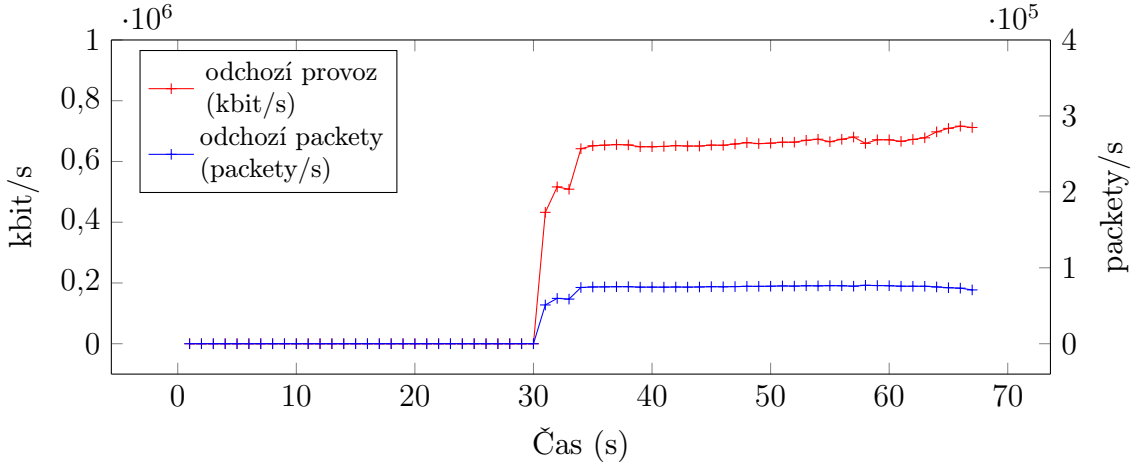


Graf A.18: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace TCP/UDP, nižší rychlost, stavová.

Filtrace TCP/UDP, vyšší rychlost, stavová



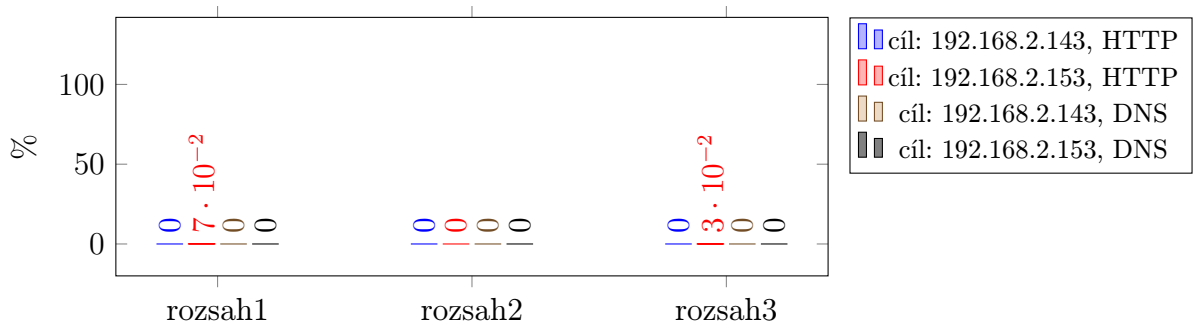
Graf A.19: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace TCP/UDP, vyšší rychlost, stavová.



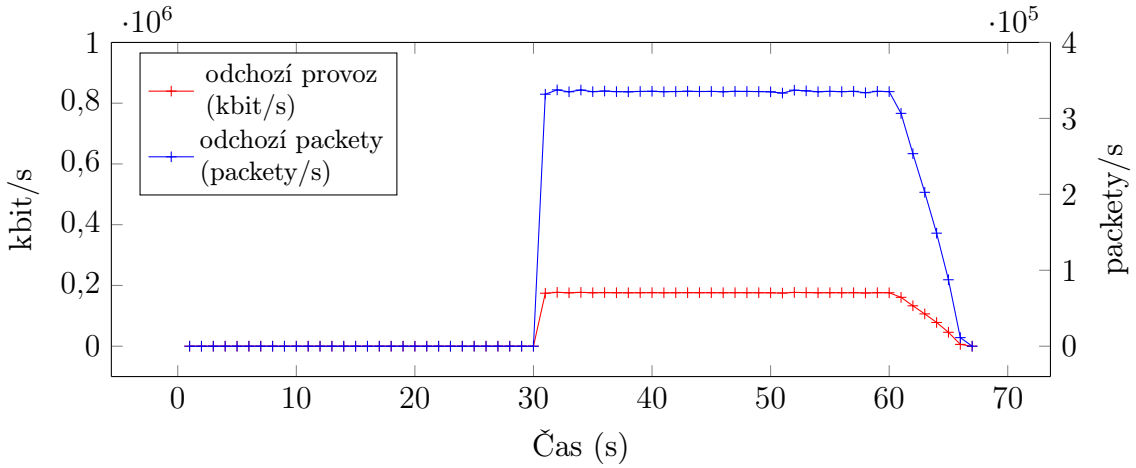
Graf A.20: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace TCP/UDP, vyšší rychlost, stavová.

A.3.2. nftables

Bez filtrace, nižší rychlost

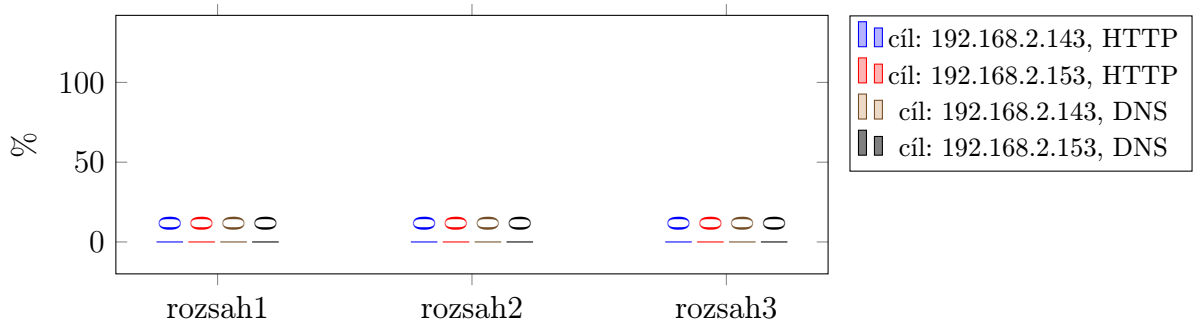


Graf A.21: Procentuální vyjádření neúspěšných transakcí: nftables – bez filtrace, nižší rychlost.

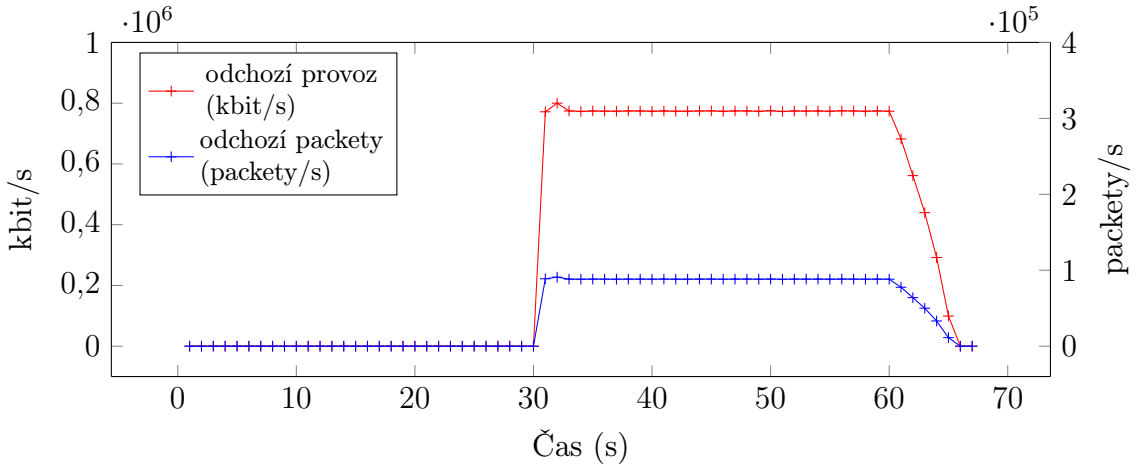


Graf A.22: Bitová a packetová rychlost v závislosti na čase: nftables – bez filtrace, nižší rychlost.

Bez filtrace, vyšší rychlost

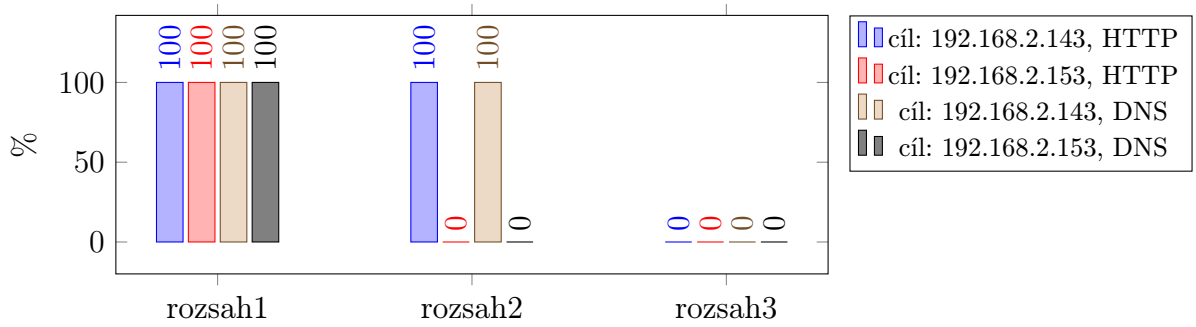


Graf A.23: Procentuální vyjádření neúspěšných transakcí: nftables – bez filtrace, vyšší rychlost.

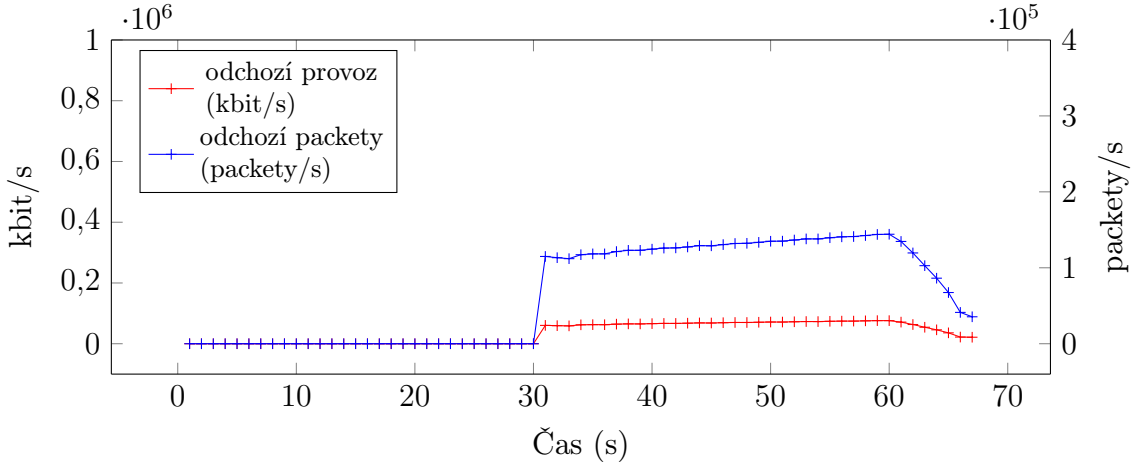


Graf A.24: Bitová a packetová rychlost v závislosti na čase: nftables – bez filtrace, vyšší rychlost.

Filtrace IP adres, nižší rychlost

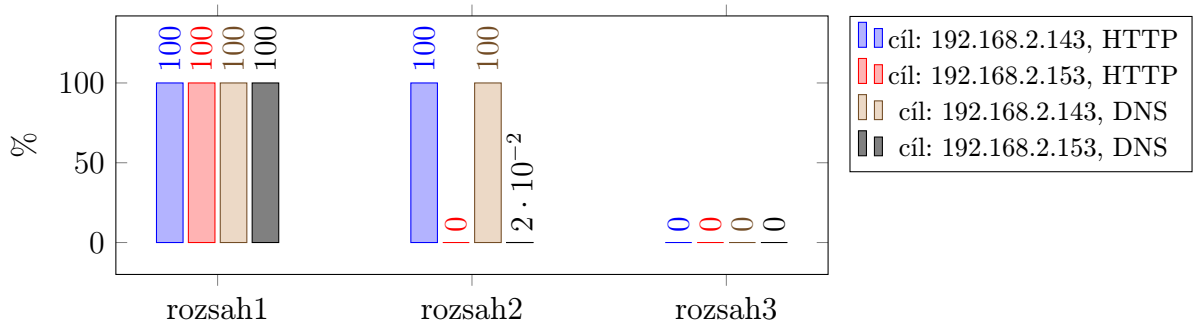


Graf A.25: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace IP adres, nižší rychlost.

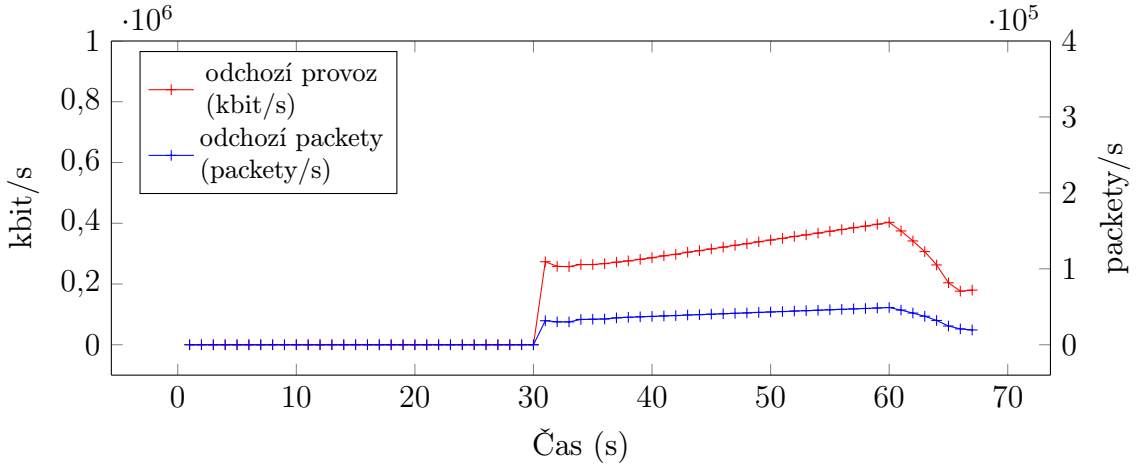


Graf A.26: Bitová a packetová rychlost v závislosti na čase: nftables – filtrace IP adres, nižší rychlost.

Filtrace IP adres, vyšší rychlost

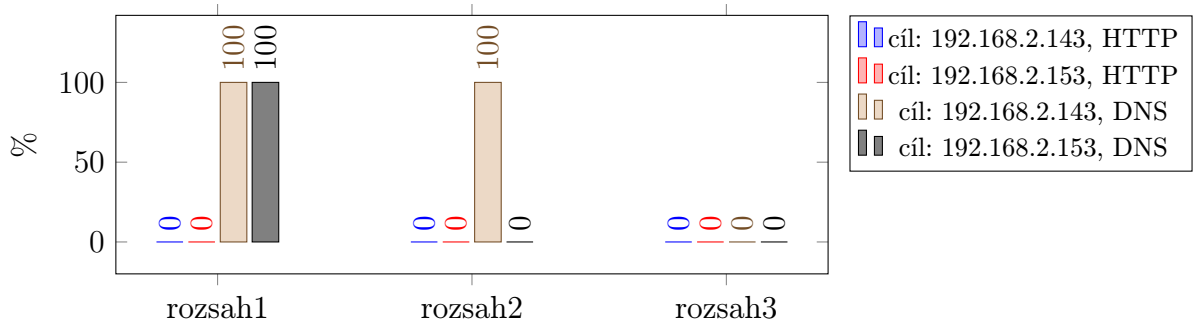


Graf A.27: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace IP adres, vyšší rychlost.

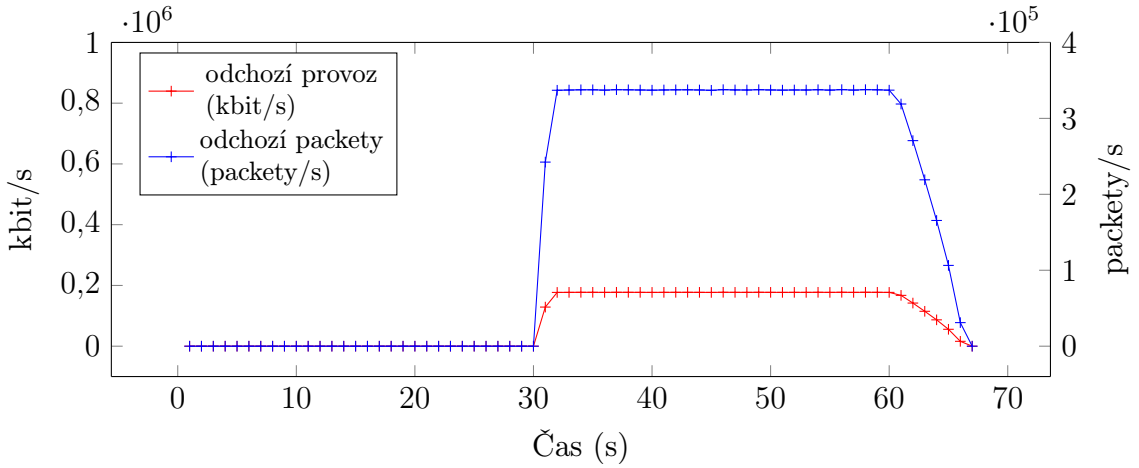


Graf A.28: Bitová a packetová rychlost v závislosti na čase: nftables – filtrace IP adres, vyšší rychlost.

Filtrace UDP portů, nižší rychlost

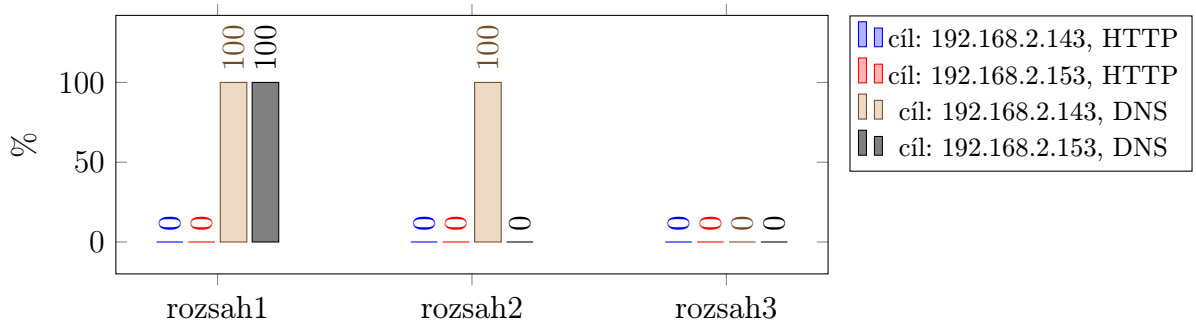


Graf A.29: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace UDP portů, nižší rychlost.

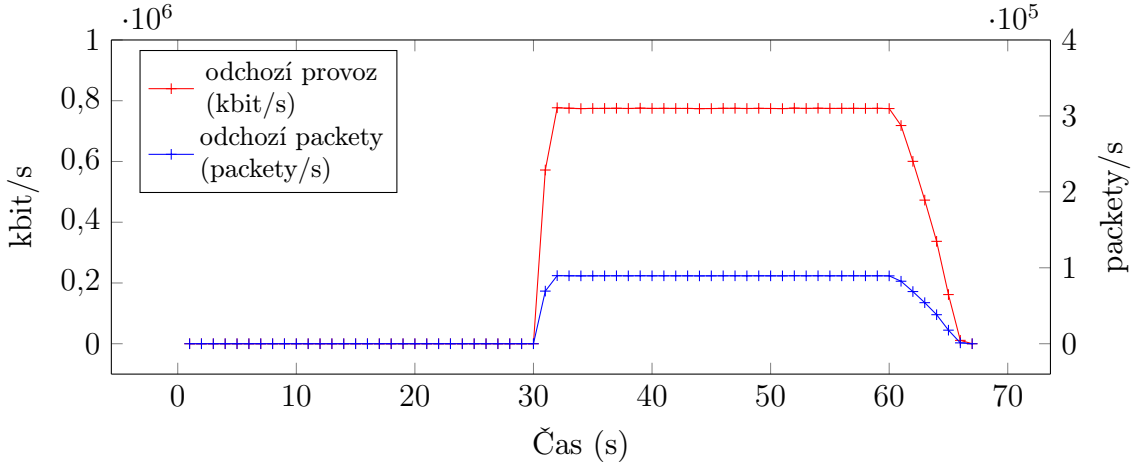


Graf A.30: Bitová a packetová rychlost v závislosti na čase: nftables – filtrace UDP portů, nižší rychlost.

Filtrace UDP portů, vyšší rychlost

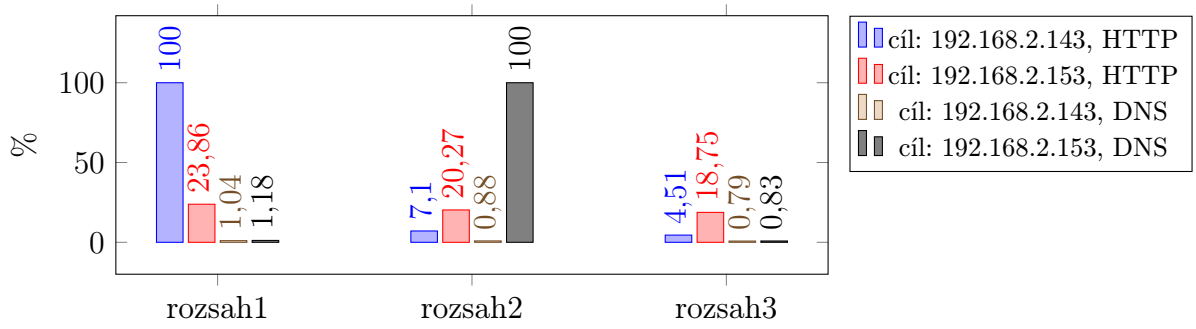


Graf A.31: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace UDP portů, vyšší rychlost.

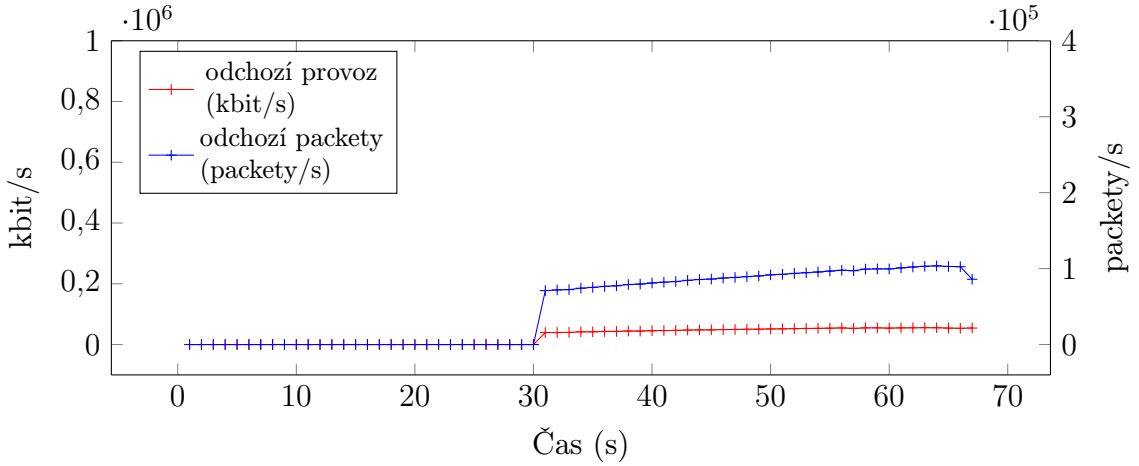


Graf A.32: Bitová a packetová rychlost v závislosti na čase: nftables – filtrace UDP portů, vyšší rychlost.

Filtrace TCP/UDP, nižší rychlost

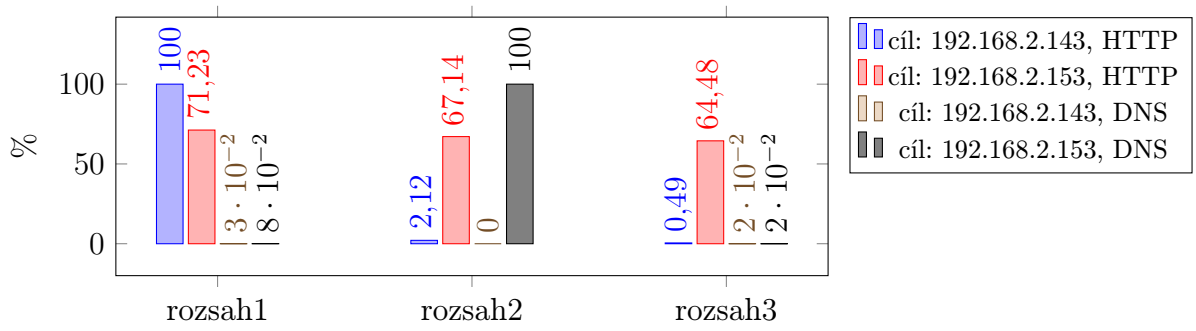


Graf A.33: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace TCP/UDP, nižší rychlost.

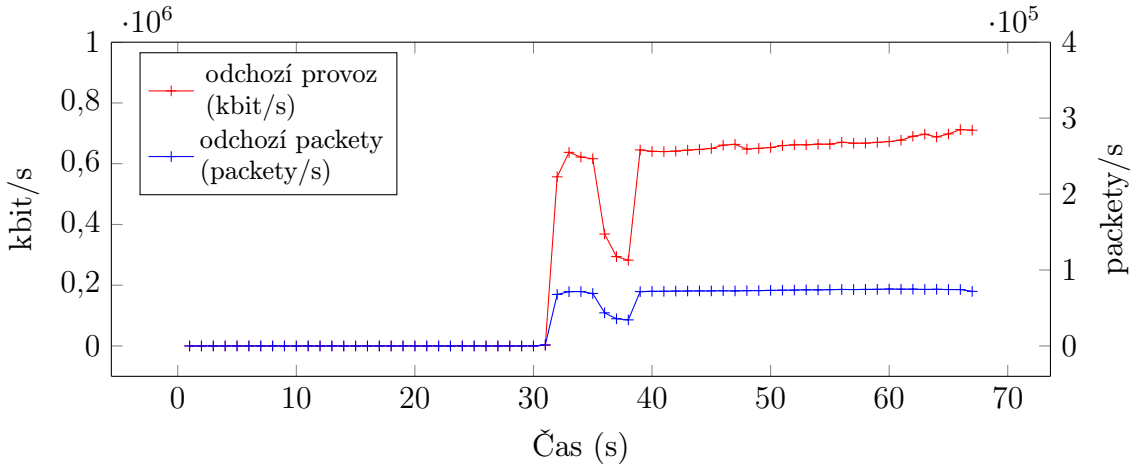


Graf A.34: Bitová a packetová rychlost v závislosti na čase: nftables – filtrace TCP/UDP, nižší rychlost.

Filtrace TCP/UDP, vyšší rychlost



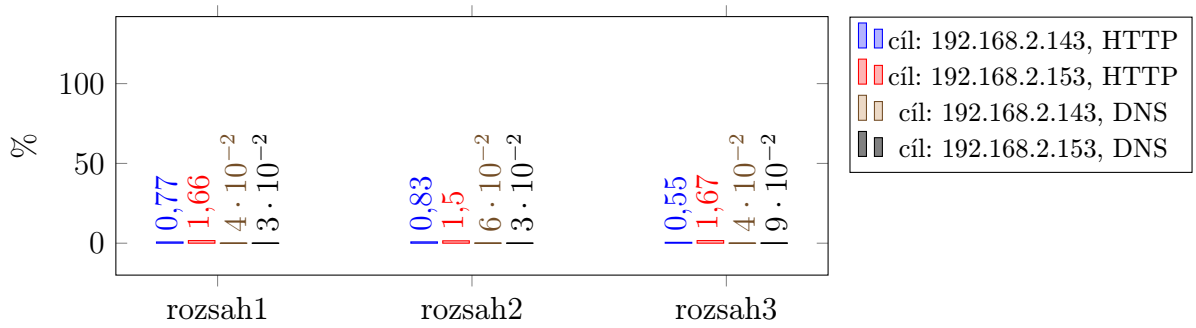
Graf A.35: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace TCP/UDP, vyšší rychlost.



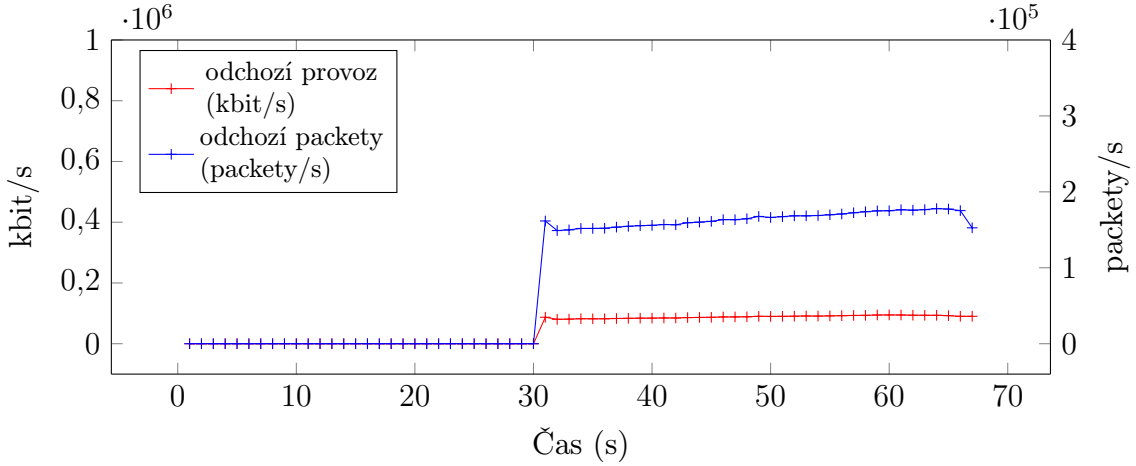
Graf A.36: Bitová a packetová rychlost v závislosti na čase: nftables – filtrace TCP/UDP, vyšší rychlost.

A.3.3. PF_RING

Bez filtrace, nižší rychlost

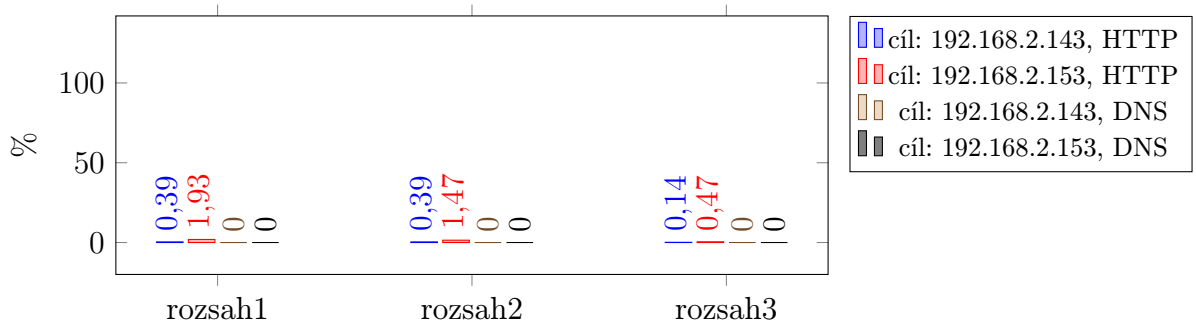


Graf A.37: Procentuální vyjádření neúspěšných transakcí: PF_RING – bez filtrace, nižší rychlost.

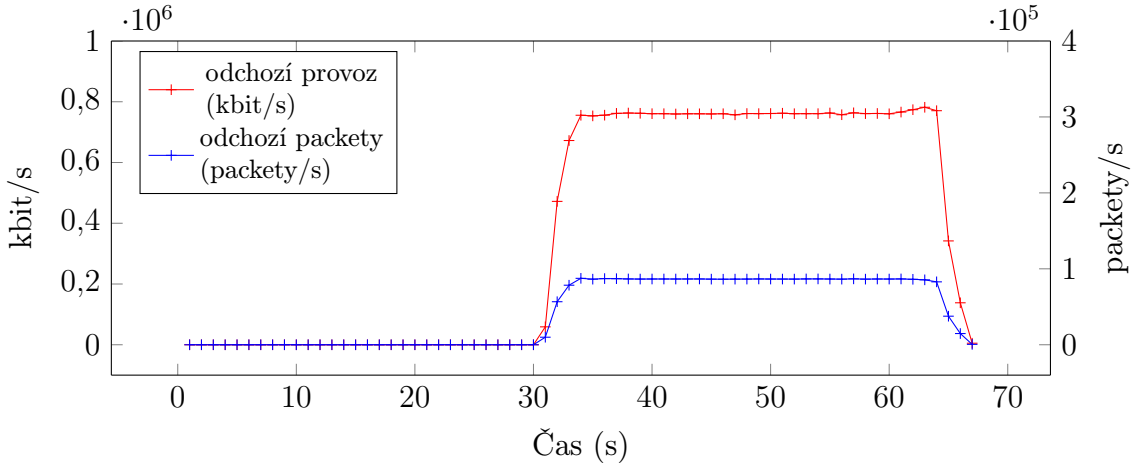


Graf A.38: Bitová a packetová rychlost v závislosti na čase: PF_RING – bez filtrace, nižší rychlost.

Bez filtrace, vyšší rychlost

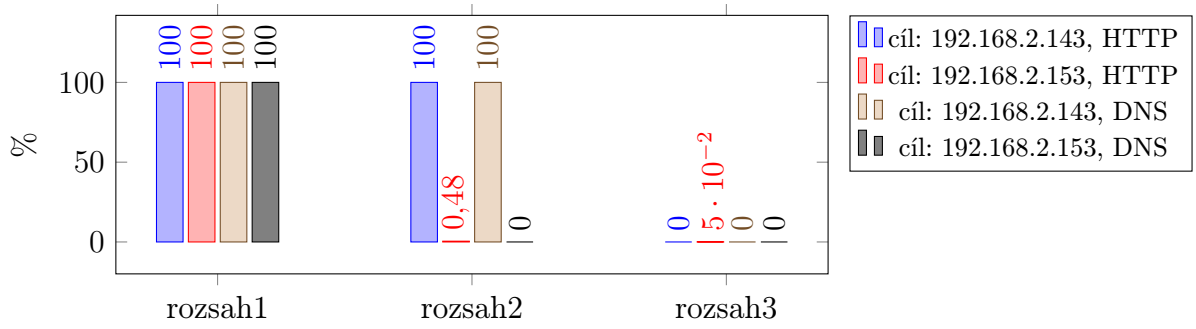


Graf A.39: Procentuální vyjádření neúspěšných transakcí: PF_RING – bez filtrace, vyšší rychlost.

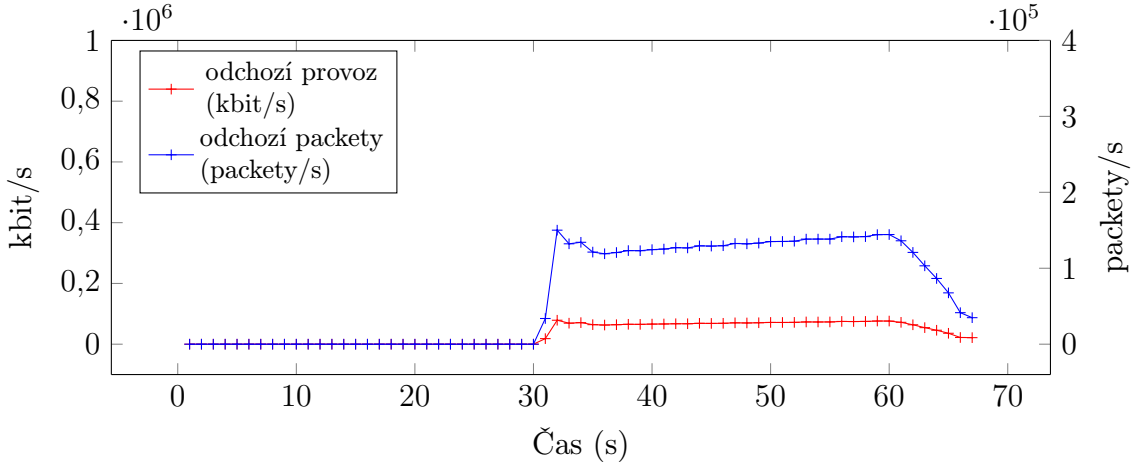


Graf A.40: Bitová a packetová rychlost v závislosti na čase: PF_RING – bez filtrace, vyšší rychlost.

Filtrace IP adres, nižší rychlost

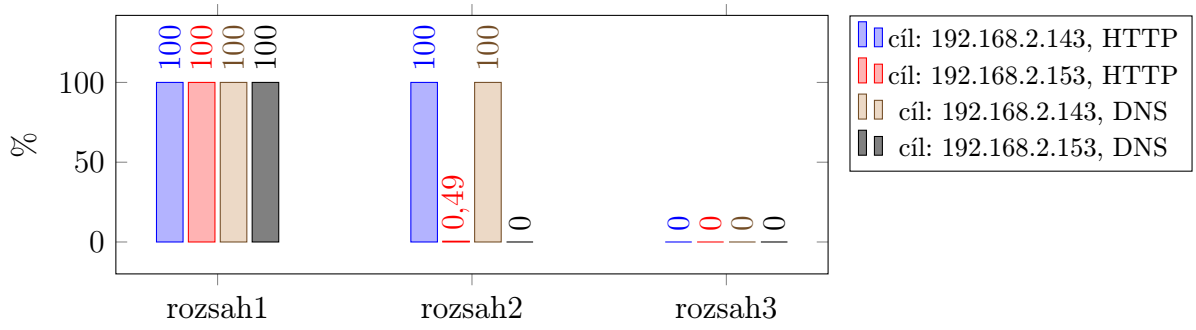


Graf A.41: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace IP adres, nižší rychlost.

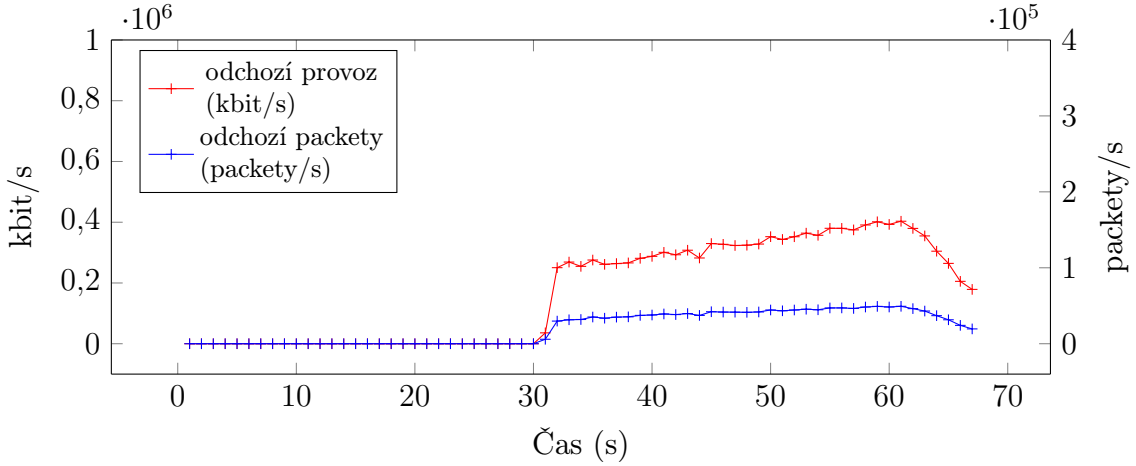


Graf A.42: Bitová a packetová rychlost v závislosti na čase: PF_RING – filtrace IP adres, nižší rychlost.

Filtrace IP adres, vyšší rychlost

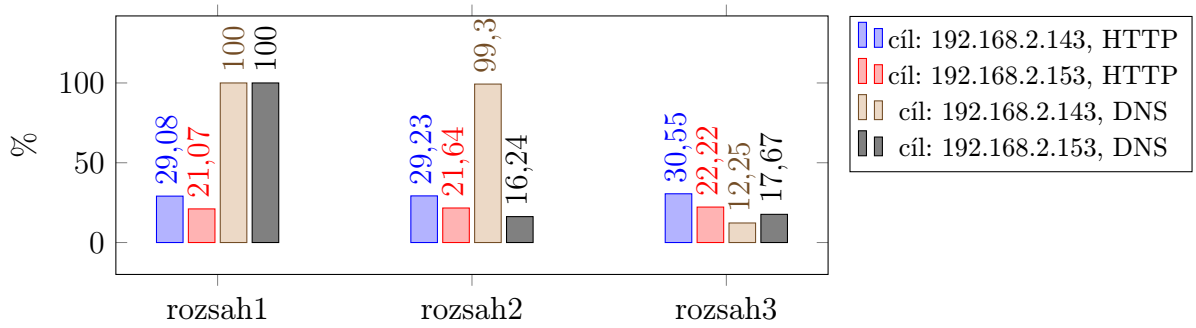


Graf A.43: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace IP adres, vyšší rychlost.

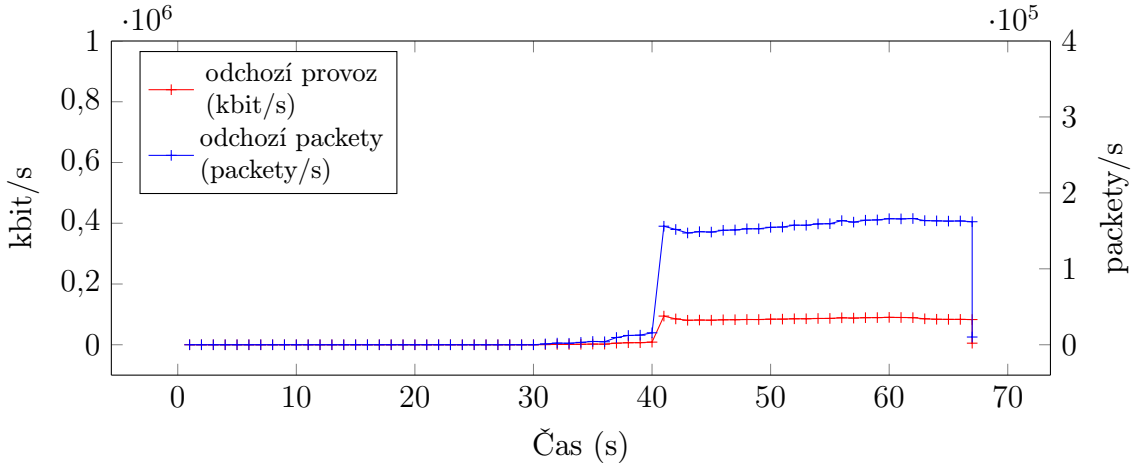


Graf A.44: Bitová a packetová rychlost v závislosti na čase: PF_RING – filtrace IP adres, vyšší rychlost.

Filtrace UDP portů, nižší rychlost

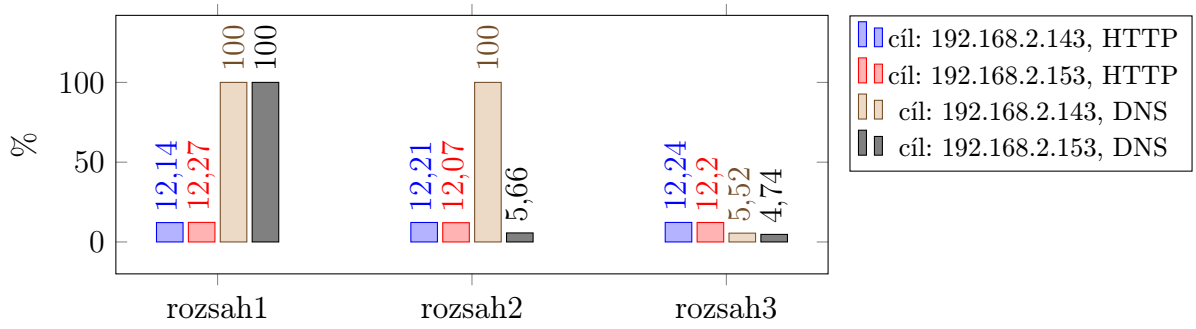


Graf A.45: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace UDP portů, nižší rychlost.

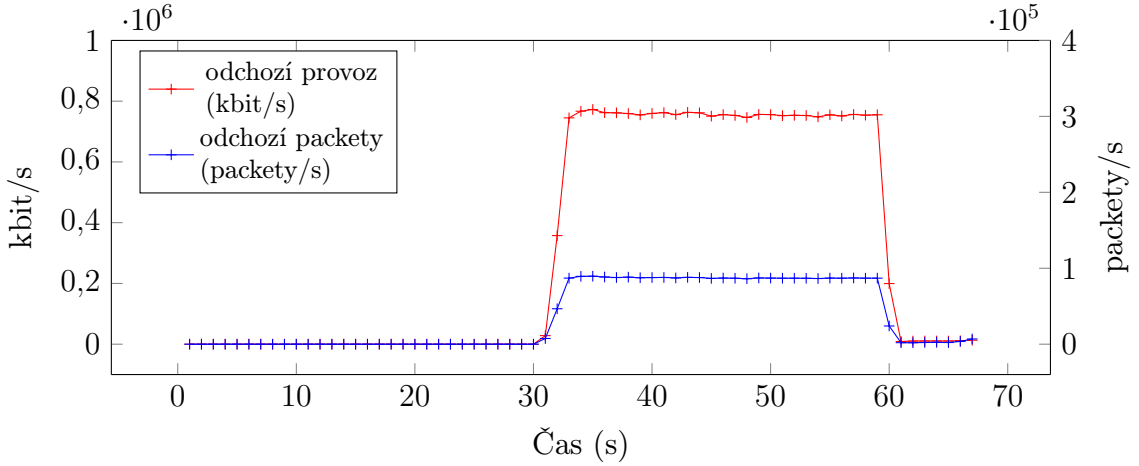


Graf A.46: Bitová a packetová rychlost v závislosti na čase: PF_RING – filtrace UDP portů, nižší rychlost.

Filtrace UDP portů, vyšší rychlost

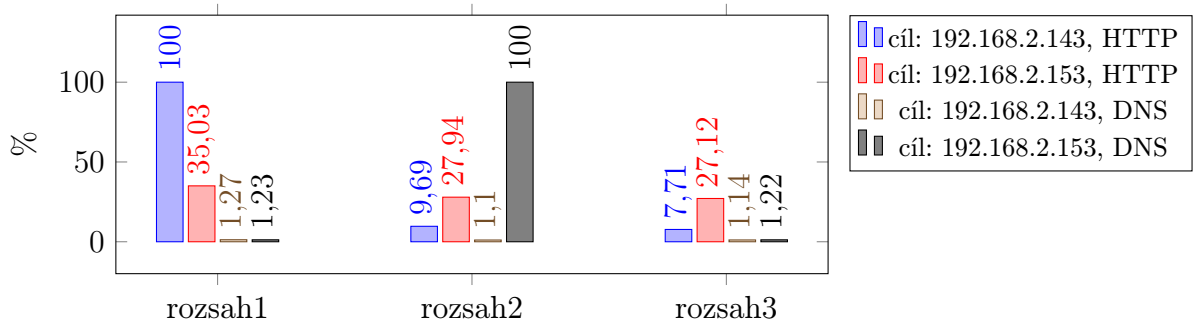


Graf A.47: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace UDP portů, vyšší rychlost.

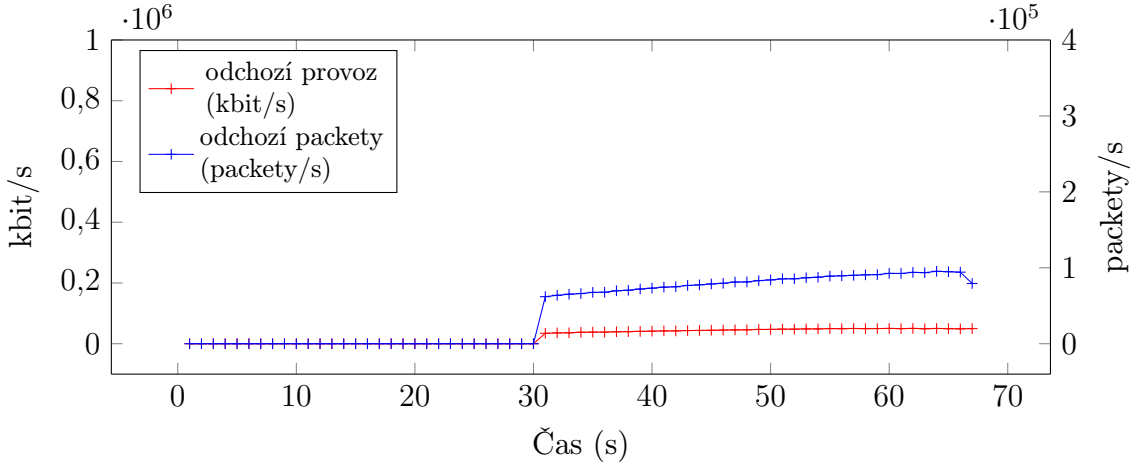


Graf A.48: Bitová a packetová rychlost v závislosti na čase: PF_RING – filtrace UDP portů, vyšší rychlost.

Filtrace TCP/UDP, nižší rychlost

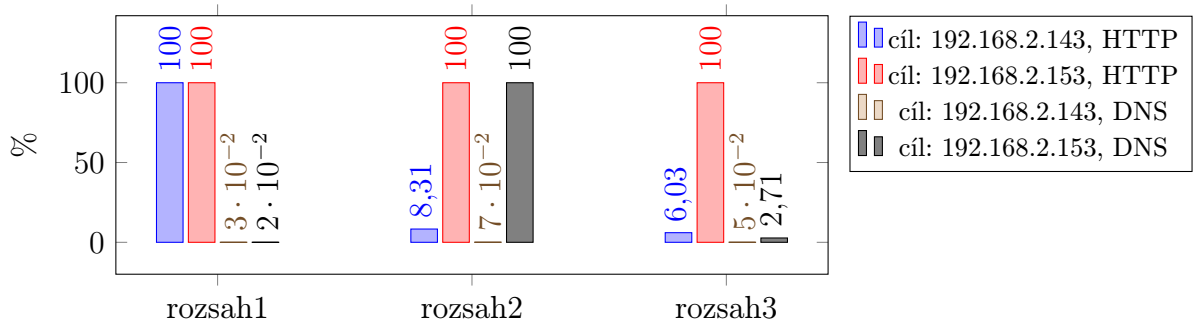


Graf A.49: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace TCP/UDP, nižší rychlost.

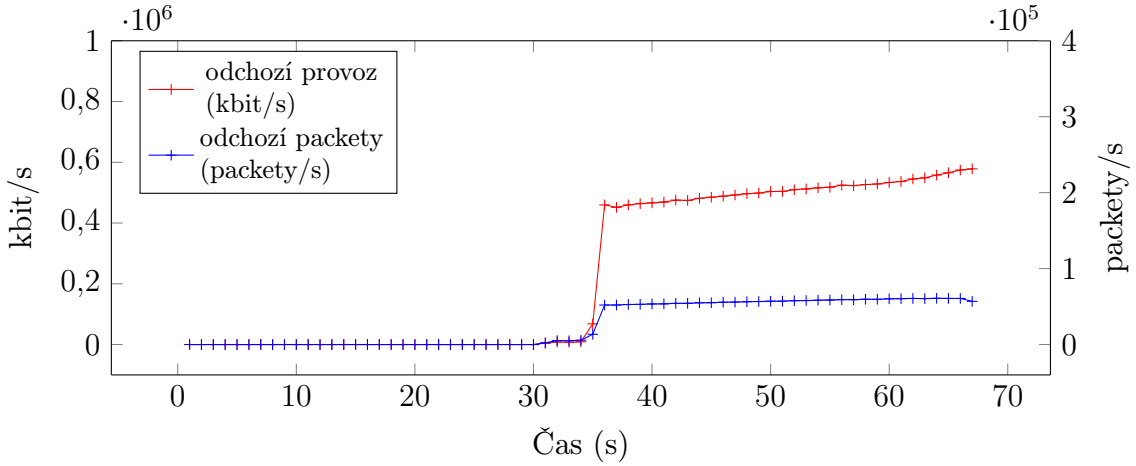


Graf A.50: Bitová a packetová rychlost v závislosti na čase: PF_RING – filtrace TCP/UDP, nižší rychlost.

Filtrace TCP/UDP, vyšší rychlost



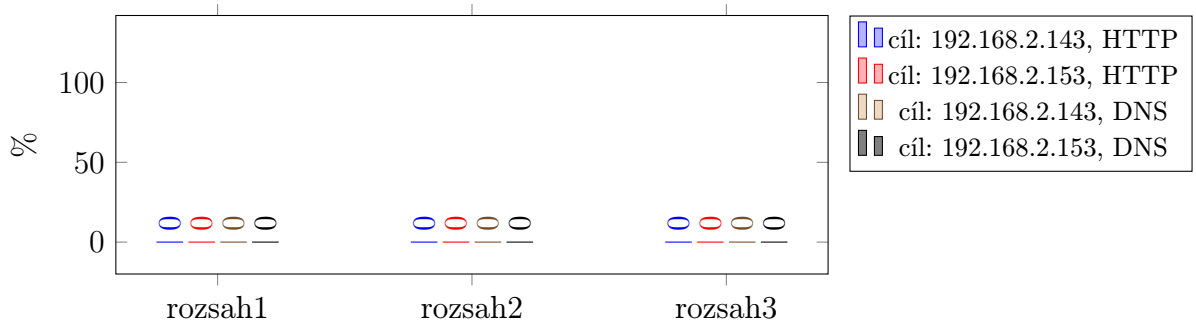
Graf A.51: Procentuální vyjádření neúspěšných transakcí: PF_RING – filtrace TCP/UDP, vyšší rychlost.



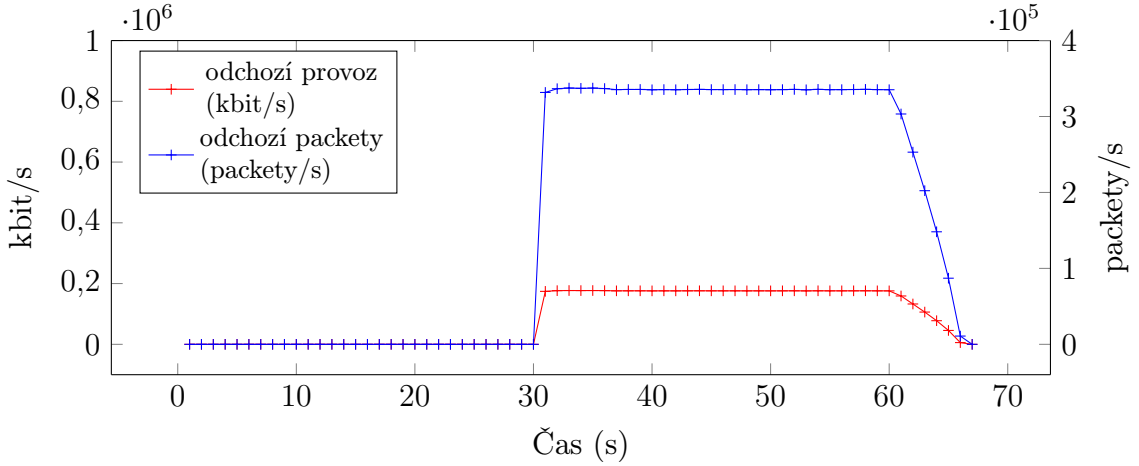
Graf A.52: Bitová a packetová rychlost v závislosti na čase: PF_RING – filtrace TCP/UDP, vyšší rychlost.

A.3.4. netmap s vlastními ovladači NIC

Bez filtrace, nižší rychlost

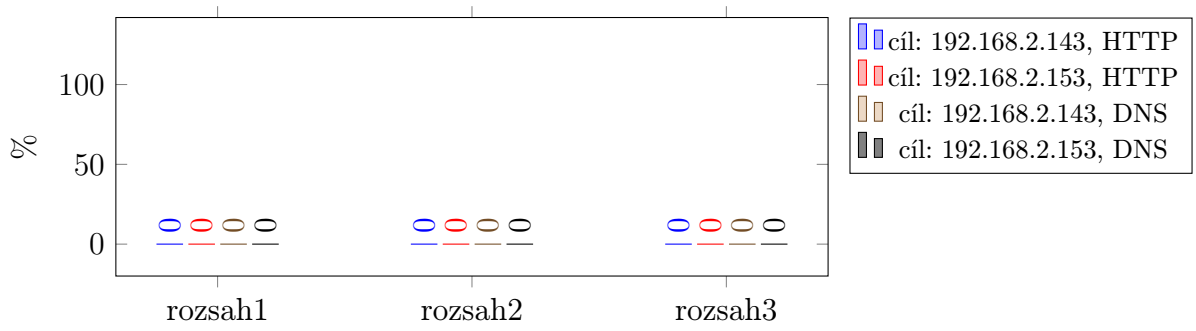


Graf A.53: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – bez filtrace, nižší rychlost.

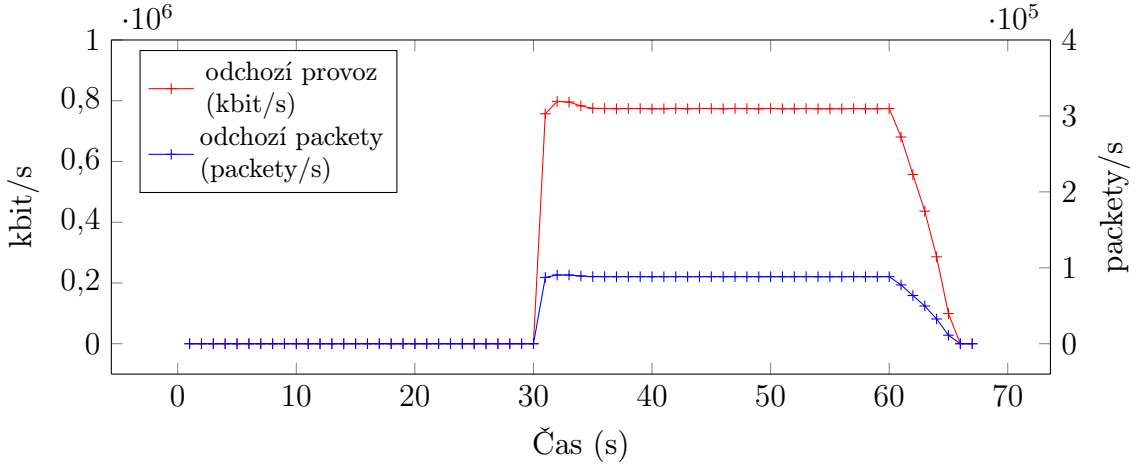


Graf A.54: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – bez filtrace, nižší rychlost.

Bez filtrace, vyšší rychlost

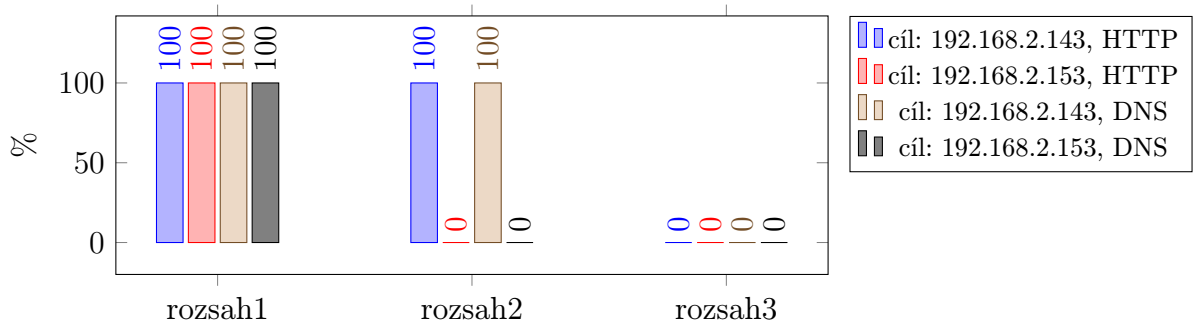


Graf A.55: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – bez filtrace, vyšší rychlost.

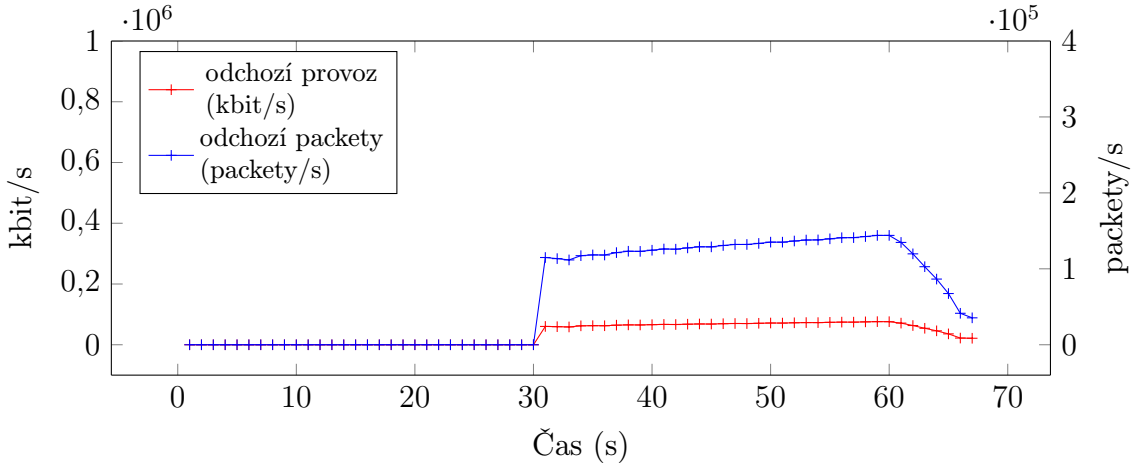


Graf A.56: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – bez filtrace, vyšší rychlost.

Filtrace IP adres, nižší rychlost

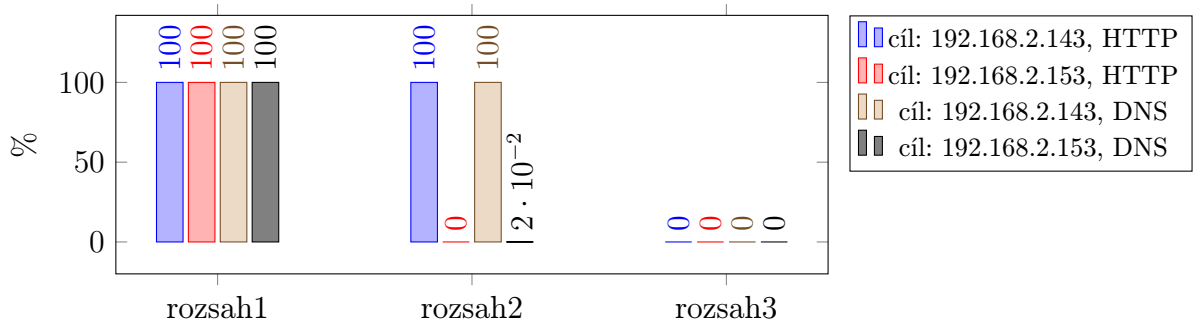


Graf A.57: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace IP adres, nižší rychlost.

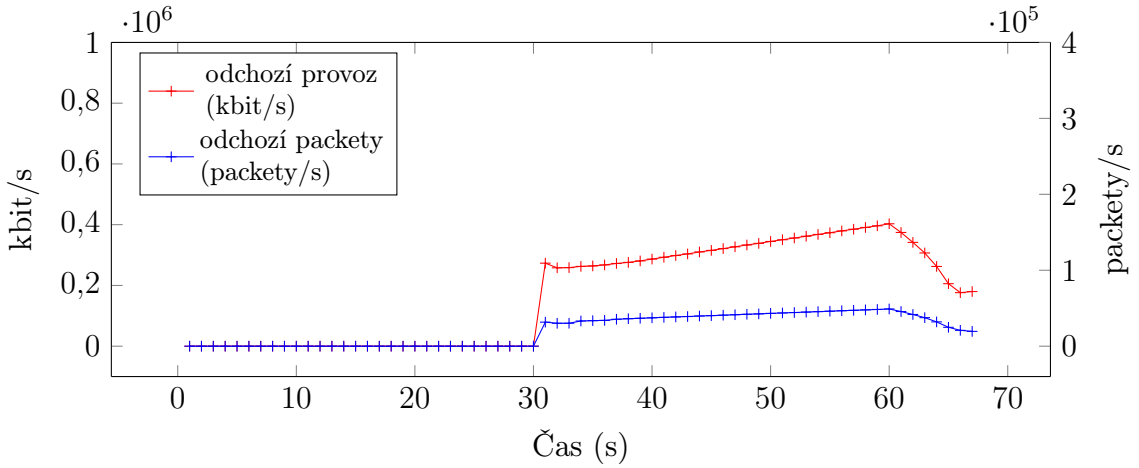


Graf A.58: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – filtrace IP adres, nižší rychlost.

Filtrace IP adres, vyšší rychlost

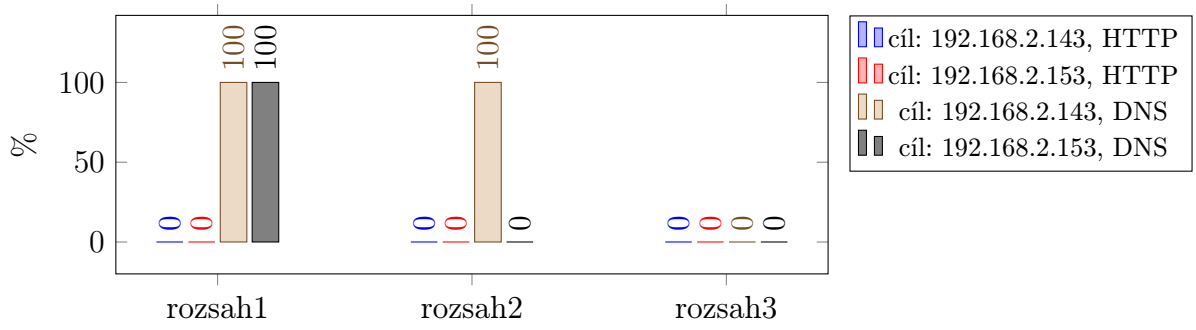


Graf A.59: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace IP adres, vyšší rychlost.

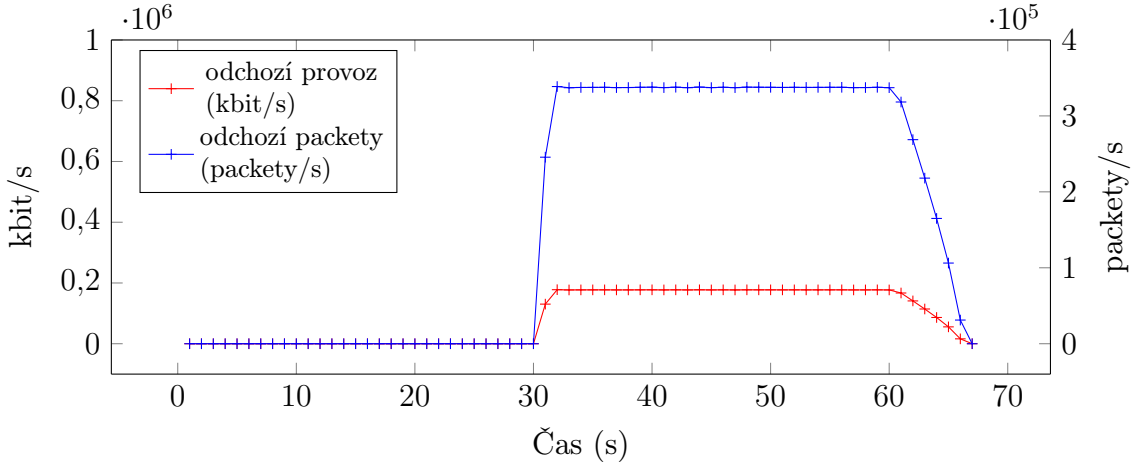


Graf A.60: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – filtrace IP adres, vyšší rychlost.

Filtrace UDP portů, nižší rychlost

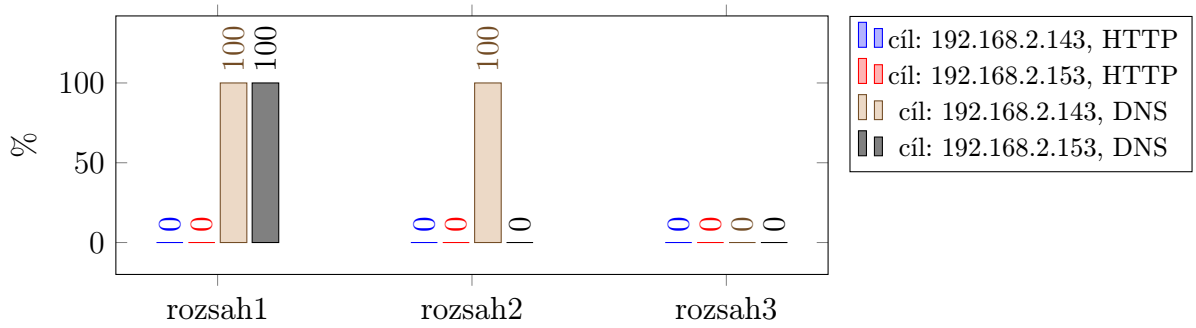


Graf A.61: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace UDP portů, nižší rychlost.

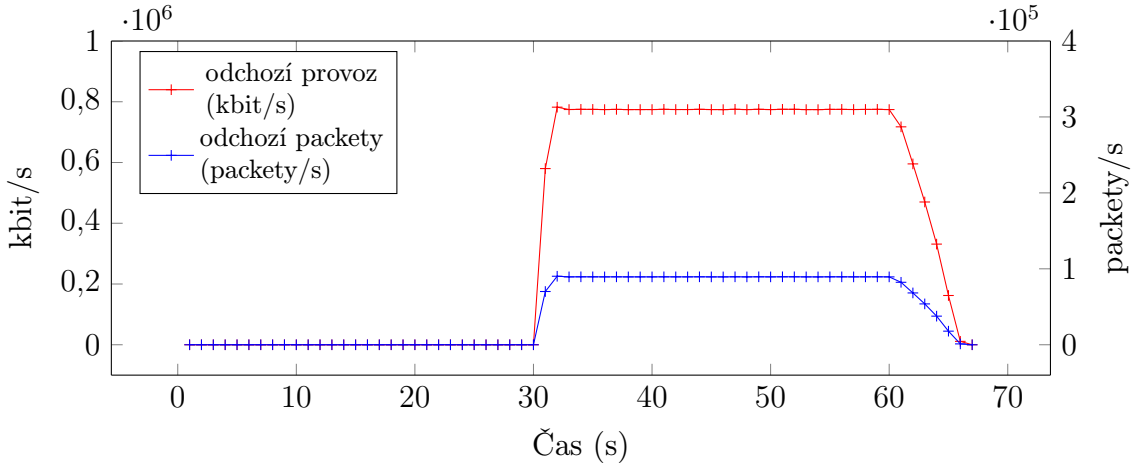


Graf A.62: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – filtrace UDP portů, nižší rychlost.

Filtrace UDP portů, vyšší rychlost

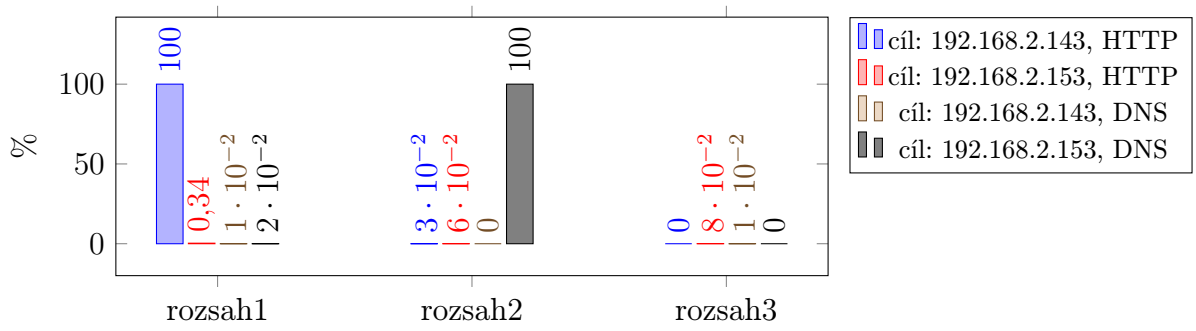


Graf A.63: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace UDP portů, vyšší rychlost.

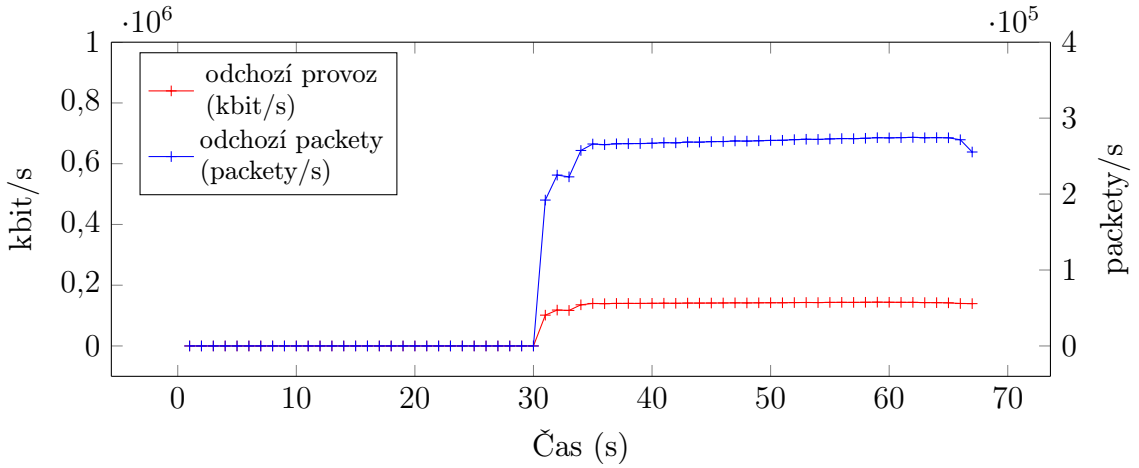


Graf A.64: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – filtrace UDP portů, vyšší rychlost.

Filtrace TCP/UDP, nižší rychlost, bezstavová

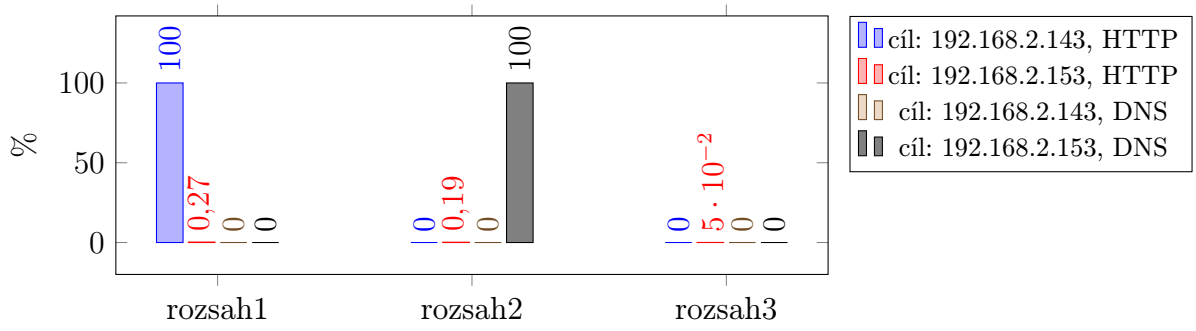


Graf A.65: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace TCP/UDP, nižší rychlost, bezstavová.

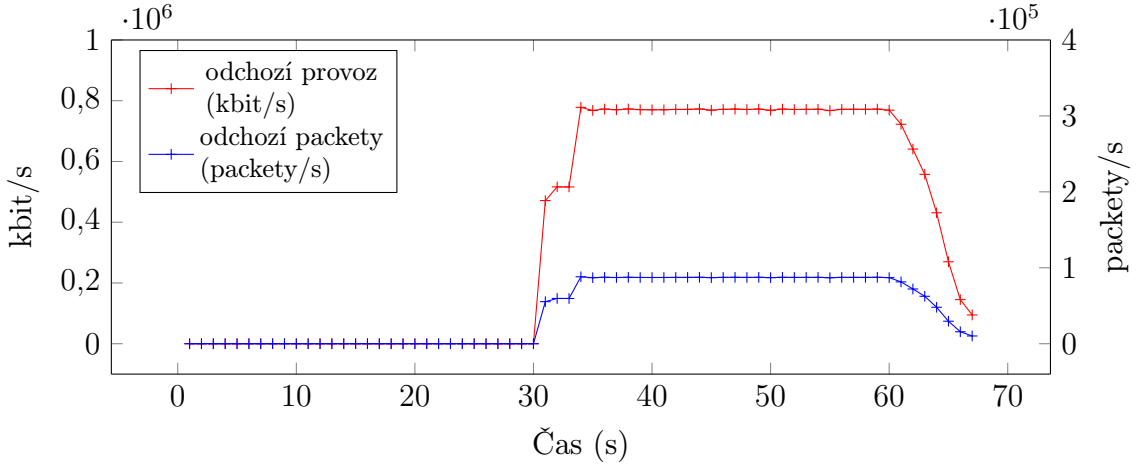


Graf A.66: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – filtrace TCP/UDP, nižší rychlost, bezstavová.

Filtrace TCP/UDP, vyšší rychlost, bezstavová

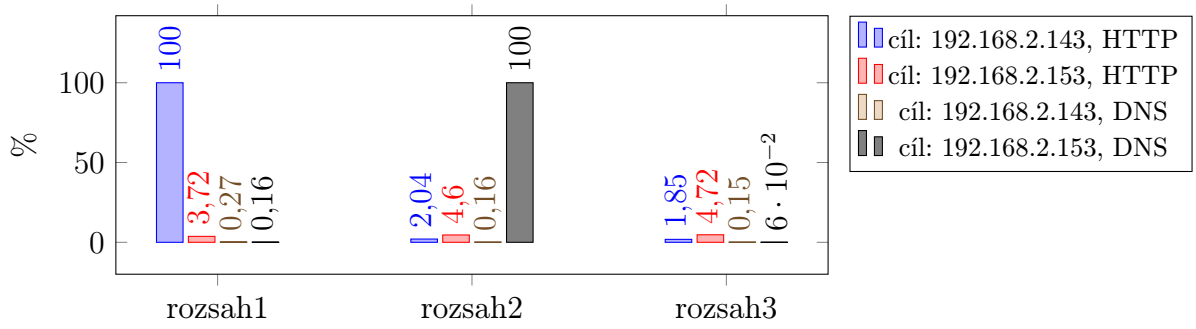


Graf A.67: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace TCP/UDP, vyšší rychlost, bezstavová.

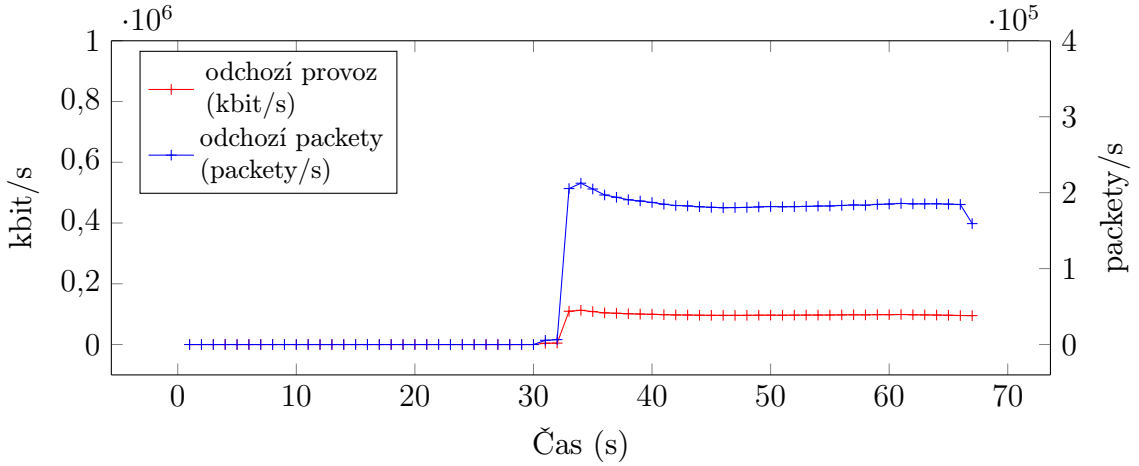


Graf A.68: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – filtrace TCP/UDP, vyšší rychlost, bezstavová.

Filtrace TCP/UDP, nižší rychlost, stavová

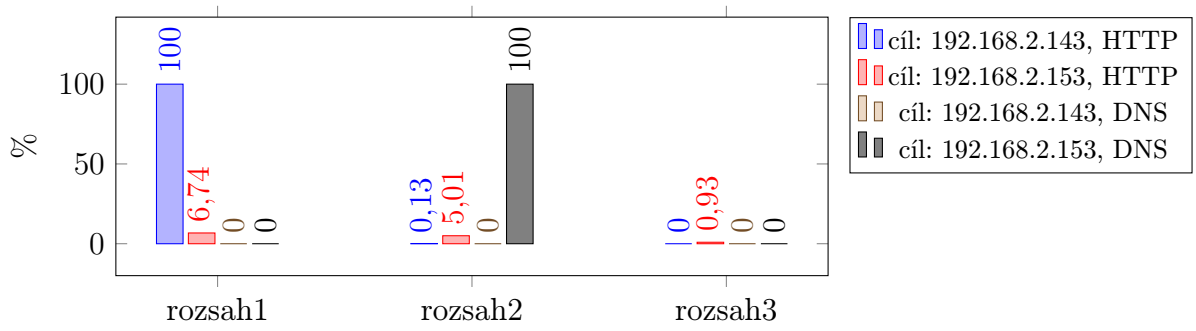


Graf A.69: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace TCP/UDP, nižší rychlost, stavová.

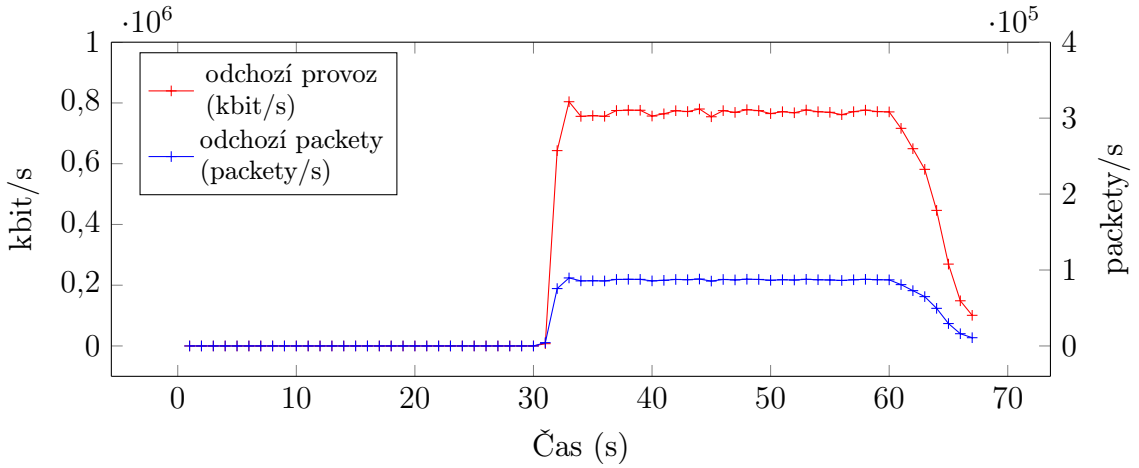


Graf A.70: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – filtrace TCP/UDP, nižší rychlost, stavová.

Filtrace TCP/UDP, vyšší rychlost, stavová



Graf A.71: Procentuální vyjádření neúspěšných transakcí: netmap (vlastní ovladač) – filtrace TCP/UDP, vyšší rychlost, stavová.

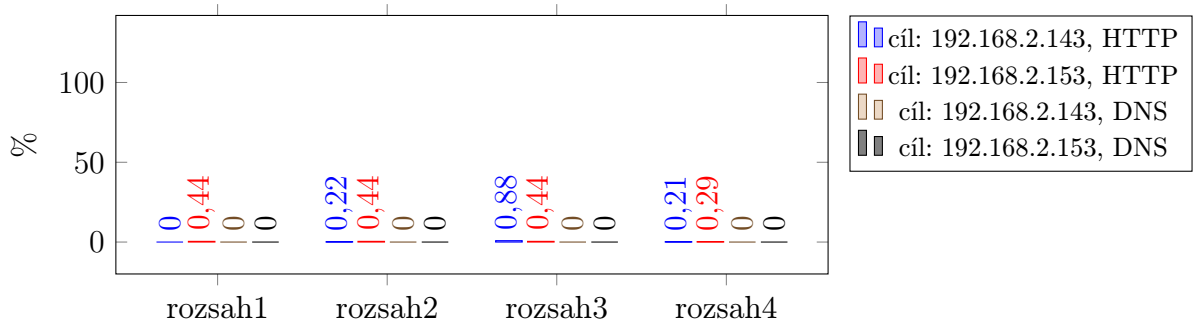


Graf A.72: Bitová a packetová rychlost v závislosti na čase: netmap (vlastní ovladač) – filtrace TCP/UDP, vyšší rychlost, stavová.

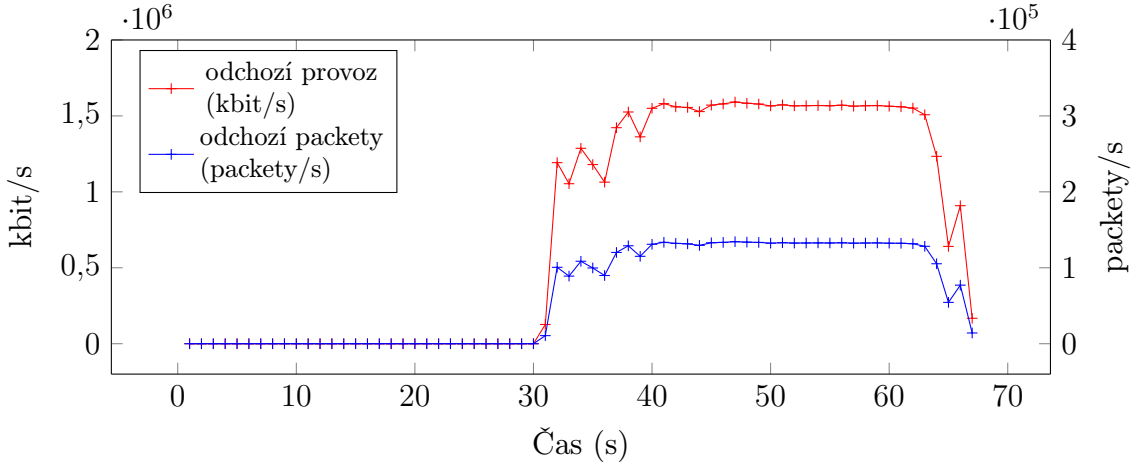
A.4. Agregace síťových rozhraní v experimentální síti

A.4.1. netfilter + iptables

Bez filtrace

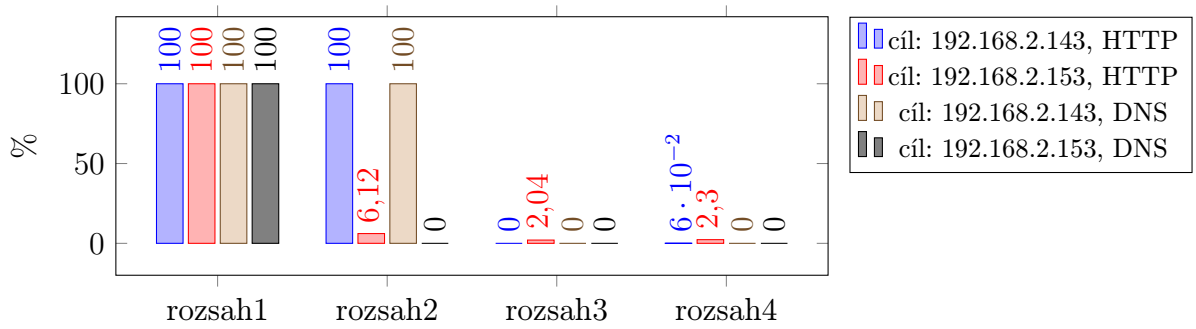


Graf A.73: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – bez filtrace.

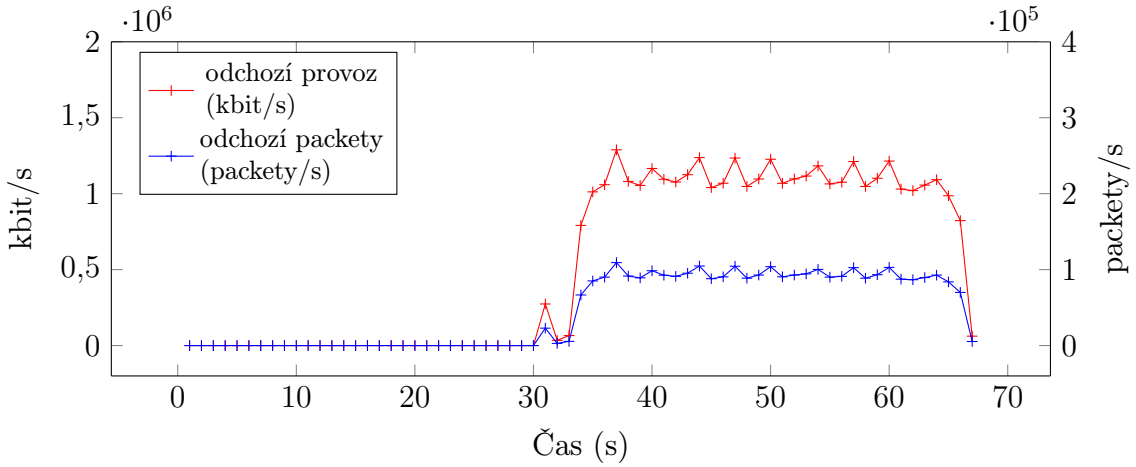


Graf A.74: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – bez filtrace.

Filtrace IP adres

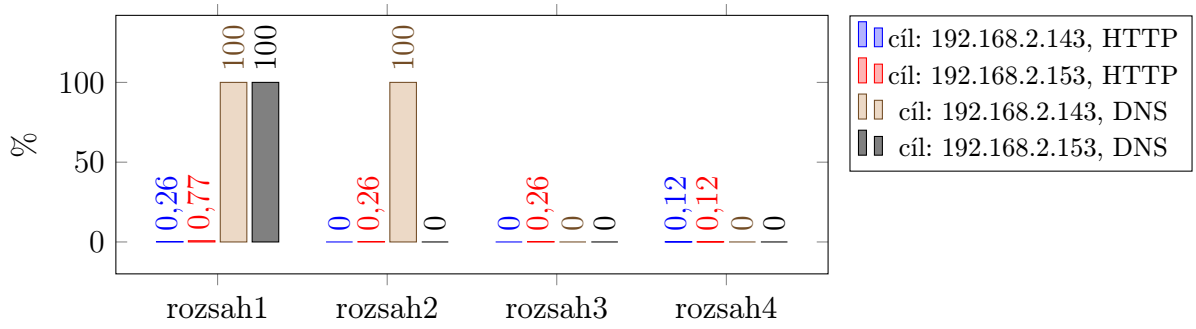


Graf A.75: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace IP adres.

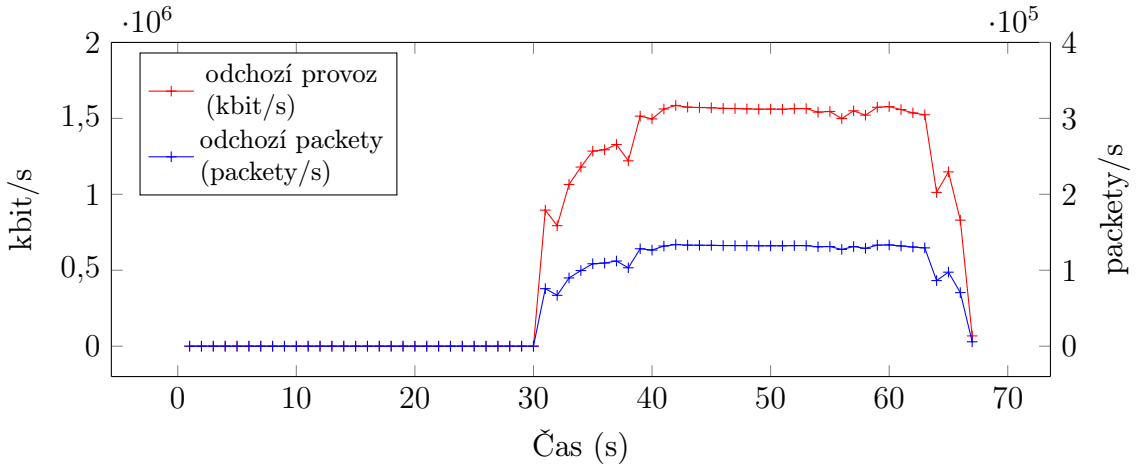


Graf A.76: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace IP adres.

Filtrace UDP portů

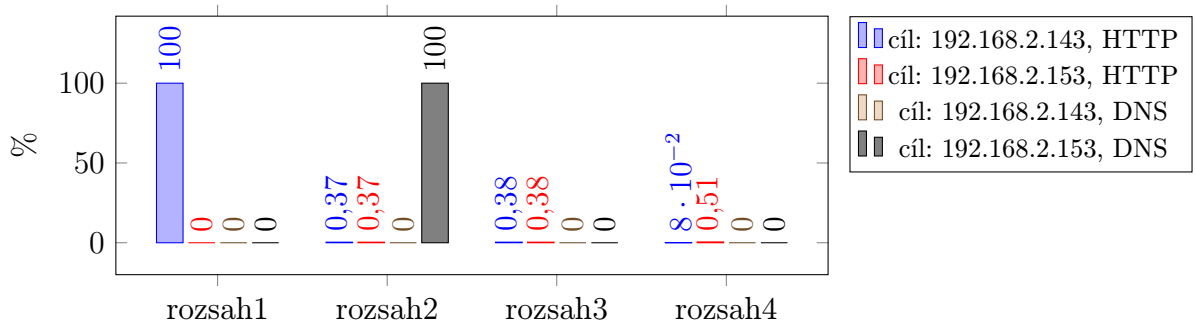


Graf A.77: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace UDP portů.

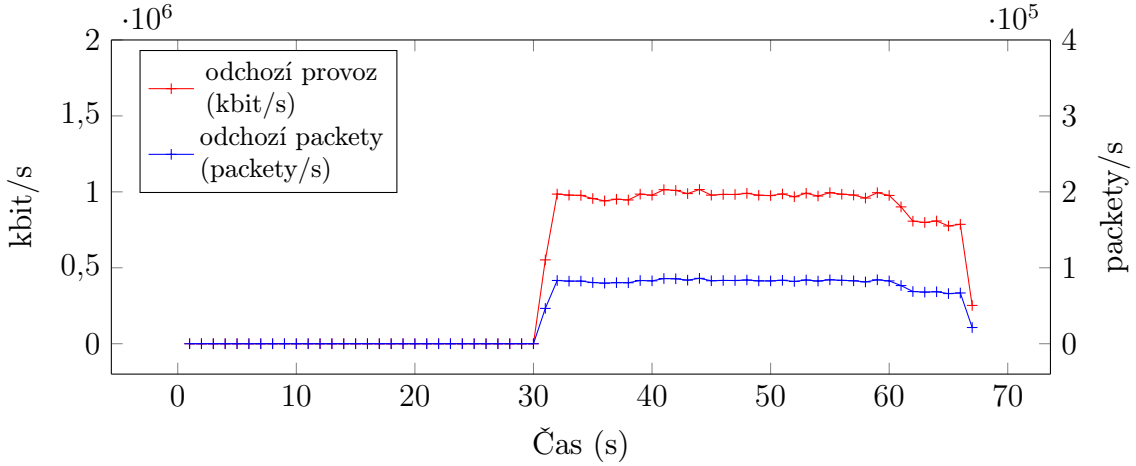


Graf A.78: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace UDP portů.

Filtrace TCP/UDP, bezstavová

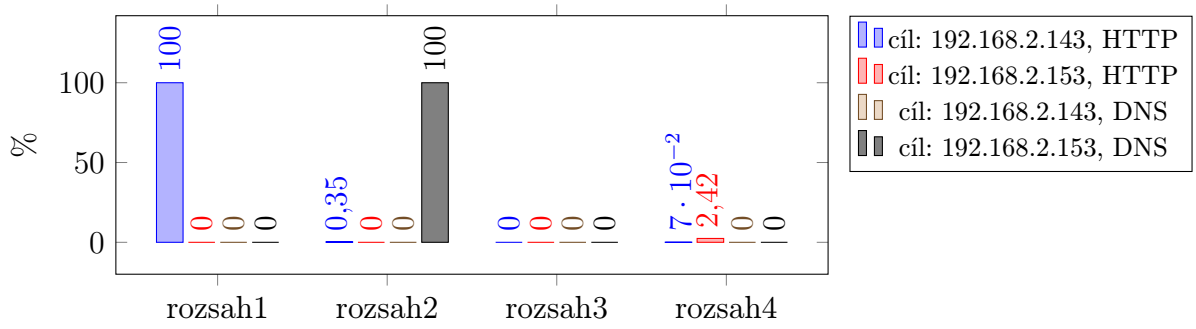


Graf A.79: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace TCP/UDP, bezstavová.

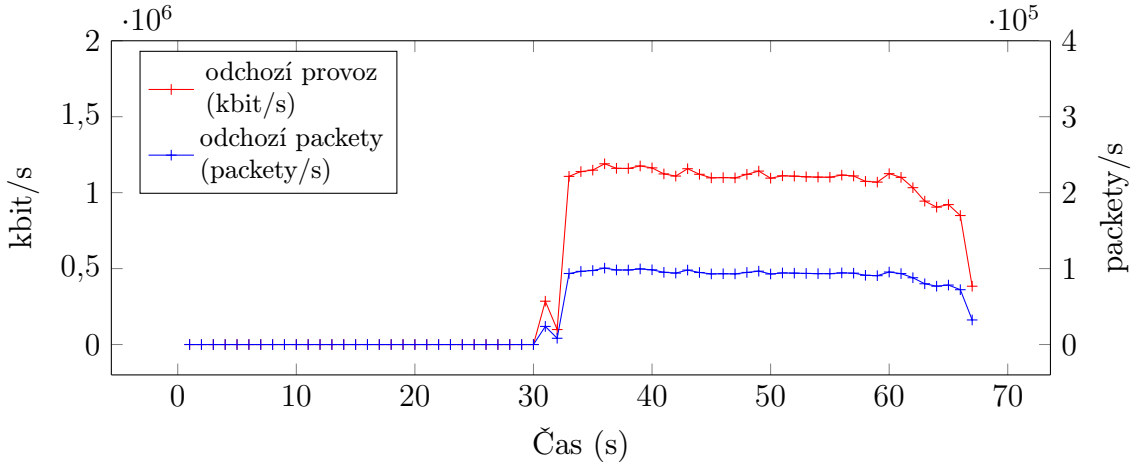


Graf A.80: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace TCP/UDP, bezstavová.

Filtrace TCP/UDP, stavová



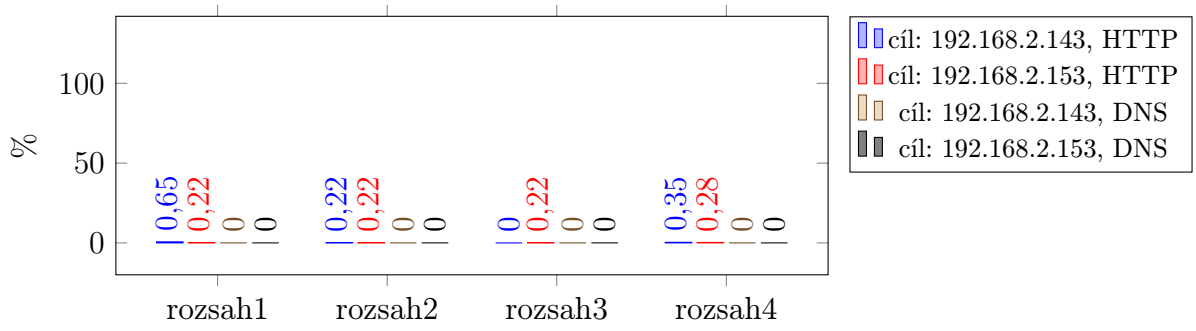
Graf A.81: Procentuální vyjádření neúspěšných transakcí: netfilter + `iptables` – filtrace TCP/UDP, stavová.



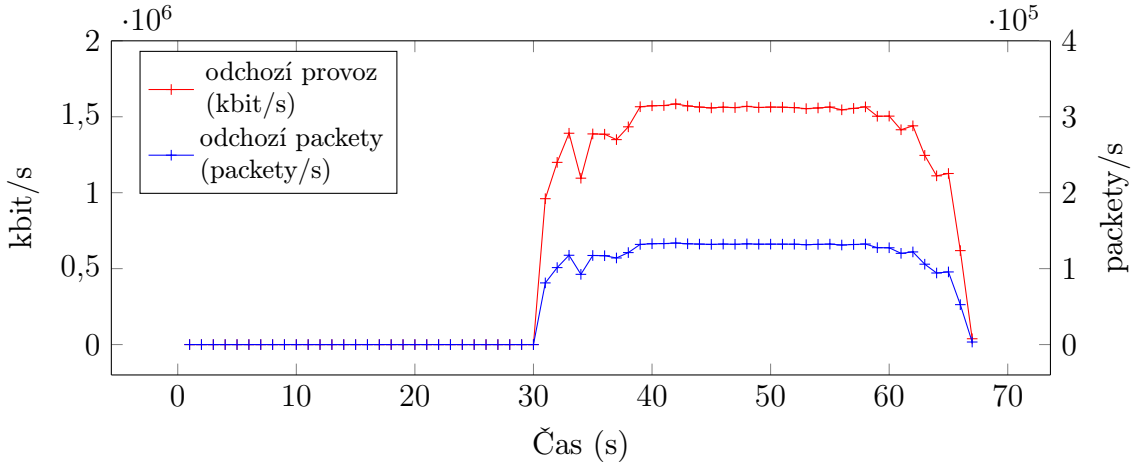
Graf A.82: Bitová a packetová rychlost v závislosti na čase: netfilter + `iptables` – filtrace TCP/UDP, stavová.

A.4.2. nftables

Bez filtrace

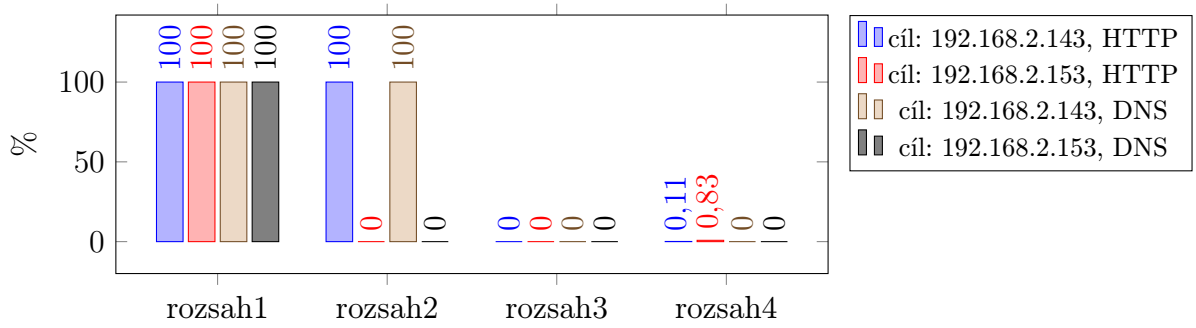


Graf A.83: Procentuální vyjádření neúspěšných transakcí: nftables – bez filtrace.

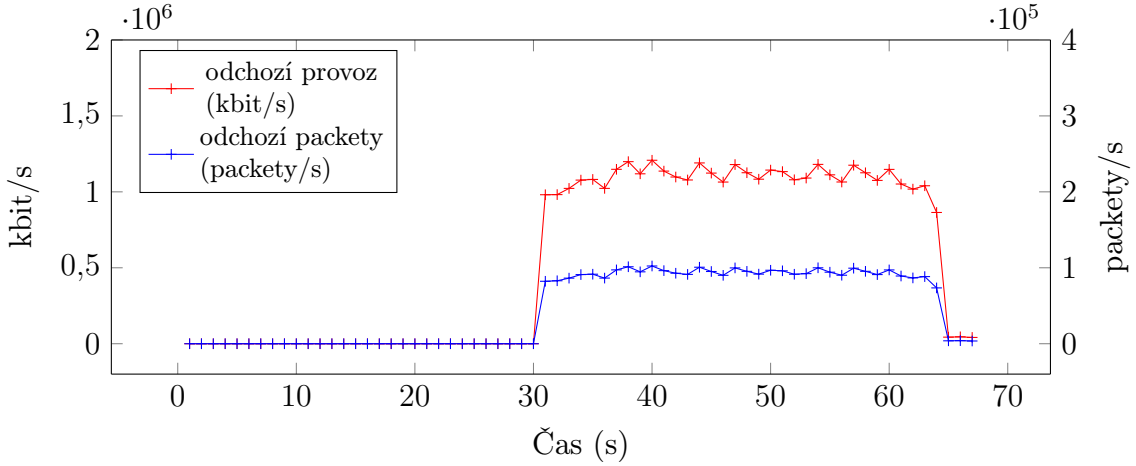


Graf A.84: Bitová a packetová rychlost v závislosti na čase: nftables – bez filtrace.

Filtrace IP adres

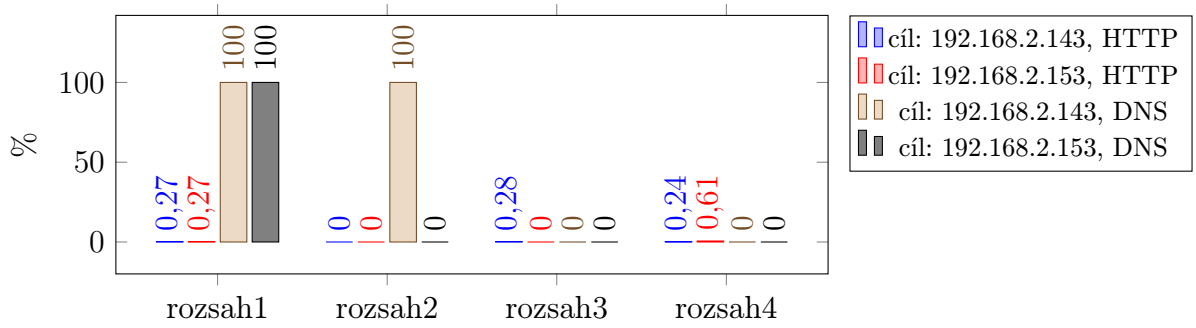


Graf A.85: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace IP adres.

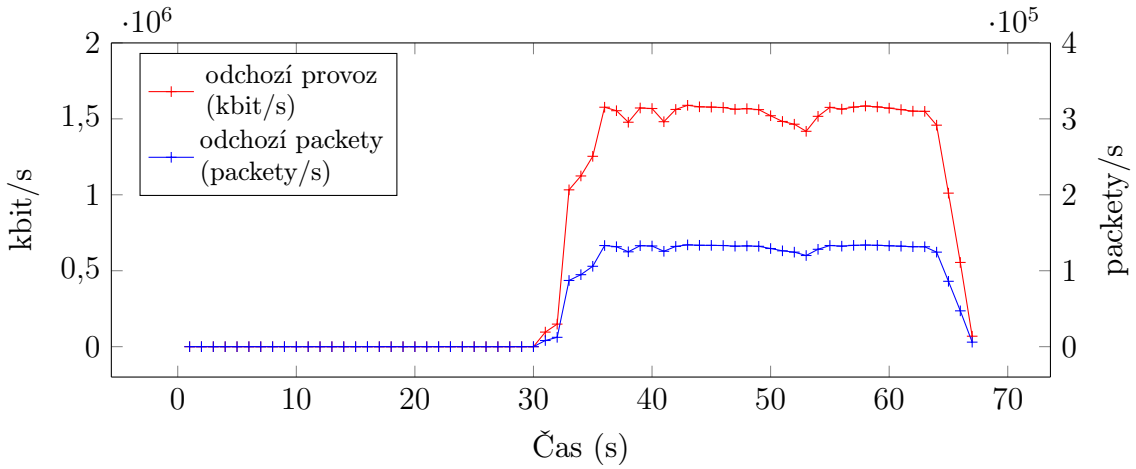


Graf A.86: Bitová a packetová rychlost v závislosti na čase: nftables – filtrace IP adres.

Filtrace UDP portů

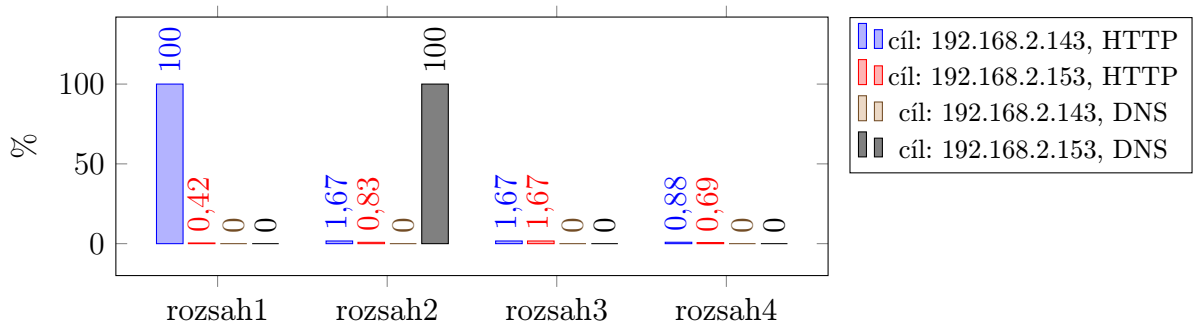


Graf A.87: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace UDP portů.

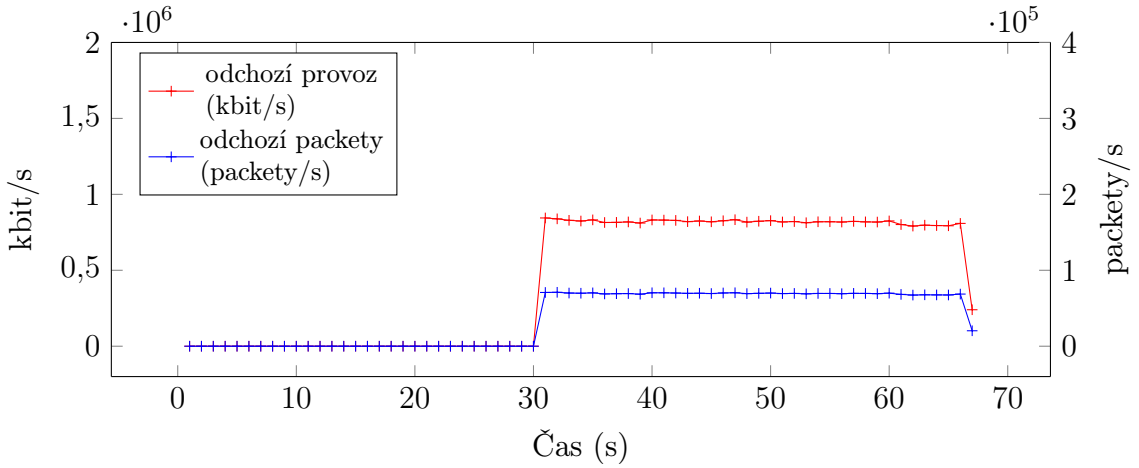


Graf A.88: Bitová a packetová rychlost v závislosti na čase: nftables – filtrace UDP portů.

Filtrace TCP/UDP



Graf A.89: Procentuální vyjádření neúspěšných transakcí: nftables – filtrace TCP/UDP.



Graf A.90: Bitová a packetová rychlost v závislosti na čase: nftables – filtrace TCP/UDP.

A.5. Příkazy pro filtraci TCP/UDP provozu nástrojem PF_RING

```
1  (term1)# ./pfbridge -a eth1 -b eth3 -f "not src host 192.168.3.1 and\
2            not src host 192.168.3.2 and not src host 192.168.3.3 and\
3            not src host 192.168.3.4 and not src host 192.168.3.5 and\
4            not src host 192.168.3.6 and not src host 192.168.3.7 and\
5            not src host 192.168.3.8 and not src host 192.168.3.9 and\
6            not src host 192.168.3.10 and not src host 192.168.3.11\
7            and not src host 192.168.3.12 and not src host 192.168.3.13\
8            and not src host 192.168.3.14 and not src host 192.168.3.15\
9            and not src host 192.168.3.16 and not src host 192.168.3.17\
10           and not src host 192.168.3.18 and not src host 192.168.3.19\
11           and not src host 192.168.3.20 and not src host 192.168.3.21\
12           and not src host 192.168.3.22 and not src host 192.168.3.23\
13           and not src host 192.168.3.24 and not src host 192.168.3.25\
14           and not src host 192.168.3.26 and not src host 192.168.3.27\
15           and not src host 192.168.3.28 and not src host 192.168.3.29\
16           and not src host 192.168.3.30 and not src host 192.168.3.31\
```

```
17            and not src host 192.168.3.32 and not src host 192.168.3.33\
18            and not src host 192.168.3.34 and not src host 192.168.3.35\
19            and not src host 192.168.3.36 and not src host 192.168.3.37\
20            and not src host 192.168.3.38 and not src host 192.168.3.39\
21            and not src host 192.168.3.40 and not src host 192.168.3.41\
22            and not src host 192.168.3.42 and not src host 192.168.3.43\
23            and not src host 192.168.3.44 and not src host 192.168.3.45\
24            and not src host 192.168.3.46 and not src host 192.168.3.47\
25            and not src host 192.168.3.48 and not src host 192.168.3.49\
26            and not src host 192.168.3.50 and not src host 192.168.3.51\
27            and not src host 192.168.3.52 and not src host 192.168.3.53\
28            and not src host 192.168.3.54 and not src host 192.168.3.55\
29            and not src host 192.168.3.56 and not src host 192.168.3.57\
30            and not src host 192.168.3.58 and not src host 192.168.3.59\
31            and not src host 192.168.3.60 and not src host 192.168.3.61\
32            and not src host 192.168.3.62 and not src host 192.168.3.63\
33            and not src host 192.168.3.64 and not src host 192.168.3.65\
34            and not src host 192.168.3.66 and not src host 192.168.3.67\
35            and not src host 192.168.3.68 and not src host 192.168.3.69\
36            and not src host 192.168.3.70 and not src host 192.168.3.71\
37            and not src host 192.168.3.72 and not src host 192.168.3.73\
38            and not src host 192.168.3.74 and not src host 192.168.3.75\
39            and not src host 192.168.3.76 and not src host 192.168.3.77\
40            and not src host 192.168.3.78 and not src host 192.168.3.79\
41            and not src host 192.168.3.80 and not src host 192.168.3.81\
42            and not src host 192.168.3.82 and not src host 192.168.3.83\
43            and not src host 192.168.3.84 and not src host 192.168.3.85\
44            and not src host 192.168.3.86 and not src host 192.168.3.87\
45            and not src host 192.168.3.88 and not src host 192.168.3.89\
46            and not src host 192.168.3.90 and not src host 192.168.3.91\
47            and not src host 192.168.3.92 and not src host 192.168.3.93\
48            and not src host 192.168.3.94 and not src host 192.168.3.95\
49            and not src host 192.168.3.96 and not src host 192.168.3.97\
50            and not src host 192.168.3.98 and not src host 192.168.3.99\
51            and not src host 192.168.3.100\
52            and not (src host 192.168.2.174 and dst host 192.168.2.143\
53            and ip proto \tcp) and not (src host 192.168.2.175\
54            and dst host 192.168.2.153 and ip proto \udp)"
55  (term2)# ./pfbridge -a eth3 -b eth1 -f "not src host 192.168.3.1 and\
56            not src host 192.168.3.2 and not src host 192.168.3.3 and\
57            not src host 192.168.3.4 and not src host 192.168.3.5 and\
58            not src host 192.168.3.6 and not src host 192.168.3.7 and\
59            not src host 192.168.3.8 and not src host 192.168.3.9 and\
60            not src host 192.168.3.10 and not src host 192.168.3.11\
61            and not src host 192.168.3.12 and not src host 192.168.3.13\
62            and not src host 192.168.3.14 and not src host 192.168.3.15\
63            and not src host 192.168.3.16 and not src host 192.168.3.17\
```

```
64              and not src host 192.168.3.18 and not src host 192.168.3.19\
65              and not src host 192.168.3.20 and not src host 192.168.3.21\
66              and not src host 192.168.3.22 and not src host 192.168.3.23\
67              and not src host 192.168.3.24 and not src host 192.168.3.25\
68              and not src host 192.168.3.26 and not src host 192.168.3.27\
69              and not src host 192.168.3.28 and not src host 192.168.3.29\
70              and not src host 192.168.3.30 and not src host 192.168.3.31\
71              and not src host 192.168.3.32 and not src host 192.168.3.33\
72              and not src host 192.168.3.34 and not src host 192.168.3.35\
73              and not src host 192.168.3.36 and not src host 192.168.3.37\
74              and not src host 192.168.3.38 and not src host 192.168.3.39\
75              and not src host 192.168.3.40 and not src host 192.168.3.41\
76              and not src host 192.168.3.42 and not src host 192.168.3.43\
77              and not src host 192.168.3.44 and not src host 192.168.3.45\
78              and not src host 192.168.3.46 and not src host 192.168.3.47\
79              and not src host 192.168.3.48 and not src host 192.168.3.49\
80              and not src host 192.168.3.50 and not src host 192.168.3.51\
81              and not src host 192.168.3.52 and not src host 192.168.3.53\
82              and not src host 192.168.3.54 and not src host 192.168.3.55\
83              and not src host 192.168.3.56 and not src host 192.168.3.57\
84              and not src host 192.168.3.58 and not src host 192.168.3.59\
85              and not src host 192.168.3.60 and not src host 192.168.3.61\
86              and not src host 192.168.3.62 and not src host 192.168.3.63\
87              and not src host 192.168.3.64 and not src host 192.168.3.65\
88              and not src host 192.168.3.66 and not src host 192.168.3.67\
89              and not src host 192.168.3.68 and not src host 192.168.3.69\
90              and not src host 192.168.3.70 and not src host 192.168.3.71\
91              and not src host 192.168.3.72 and not src host 192.168.3.73\
92              and not src host 192.168.3.74 and not src host 192.168.3.75\
93              and not src host 192.168.3.76 and not src host 192.168.3.77\
94              and not src host 192.168.3.78 and not src host 192.168.3.79\
95              and not src host 192.168.3.80 and not src host 192.168.3.81\
96              and not src host 192.168.3.82 and not src host 192.168.3.83\
97              and not src host 192.168.3.84 and not src host 192.168.3.85\
98              and not src host 192.168.3.86 and not src host 192.168.3.87\
99              and not src host 192.168.3.88 and not src host 192.168.3.89\
100             and not src host 192.168.3.90 and not src host 192.168.3.91\
101             and not src host 192.168.3.92 and not src host 192.168.3.93\
102             and not src host 192.168.3.94 and not src host 192.168.3.95\
103             and not src host 192.168.3.96 and not src host 192.168.3.97\
104             and not src host 192.168.3.98 and not src host 192.168.3.99\
105             and not src host 192.168.3.100\
106             and not (src host 192.168.2.174 and dst host 192.168.2.143\
107             and ip proto \tcp) and not (src host 192.168.2.175\
108             and dst host 192.168.2.153 and ip proto \udp)"
```